Claes Wohlin · Per Runeson ·
Martin Höst · Magnus C. Ohlsson ·
Björn Regnell · Anders Wesslén

Experimentation in Software Engineering

Second Edition

Springer

Experimentation in Software Engineering

Claes Wohlin • Per Runeson • Martin Höst •
Magnus C. Ohlsson • Björn Regnell •
Anders Wesslén

Experimentation in Software Engineering

Second Edition

Springer

Claes Wohlin
Blekinge Institute of Technology
Karlskrona, Sweden

Martin Höst
Faculty of Technology and Society
Malmö University
Malmö, Sweden

Björn Regnell
Department of Computer Science
Lund University
Lund, Sweden

Per Runeson
Department of Computer Science
Lund University
Lund, Sweden

Magnus C. Ohlsson
System Verification Sweden AB
Malmö, Sweden

Anders Wesslén
Ericsson AB
Lund, Sweden

ISBN 978-3-662-69305-6 ISBN 978-3-662-69306-3 (eBook)
https://doi.org/10.1007/978-3-662-69306-3

Endorsements

I recommend this seminal book to everyone seeking guidance in systematically carrying out software engineering experiments. My PhD students love it.
Prof. Kurt Schneider, Leibniz Universität Hannover, Germany

As a student, this book shaped my understanding of empirical software engineering, and its comprehensive, condensed wisdom now serves as a timeless guide for my students.
Dr. Maleknaz Nayebi, York University, Toronto, Canada

This book is a landmark in allowing us to train both the researcher and practitioner in software engineering experimentation.
Prof. Emeritus Victor R. Basili, University of Maryland, USA

Foreword from the First Edition

Experimentation is fundamental to any scientific and engineering endeavor.

Understanding a discipline involves building models of the various elements of the discipline, e.g., the objects in the domain, the processes used to manipulate those objects, the relationship between the processes and the objects. Evolving domain knowledge implies the evolution of those models by testing them via experiments of various forms. Analyzing the results of the experiment involves learning, the encapsulation of knowledge and the ability to change and refine our models over time. Therefore, our understanding of a discipline evolves over time.

This is the paradigm that has been used in many fields, e.g., physics, medicine, manufacturing. These fields evolved as disciplines when they began applying the cycle of model building, experimenting, and learning. Each field began with recording observations and evolved to manipulating the model variables and studying the effects of changes in those variables. The fields differ in their nature, what constitutes the basic objects of the field, the properties of those objects, the properties of the system that contain them, the relationship of the objects to the system, and the culture of the discipline. These differences affect how the models are built and how experimentation gets done.

Like other science and engineering disciplines, software engineering requires the cycle of model building, experimentation, and learning. The study of software engineering is a laboratory science. The players in the discipline are the researchers and the practitioners. The researcher's role is to understand the nature of the object (products), the processes that create and manipulate them, and the relationship between the two in the context of the system. The practitioner's role is to build 'improved' systems, using the knowledge available to date. These roles are symbiotic. The researcher needs laboratories to study the problems faced by practitioners and develop and evolve solutions based upon experimentation. The practitioner needs to understand how to build better systems and the researcher can provide models to help.

In developing models and experimenting, both the researcher and the practitioner need to understand the nature of the discipline of software engineering. All software is not the same: there are a large number of variables that cause differences and their

effects need to be understood. Like medicine where the variation in human genetics and medical history is often a major factor in the development of the models and the interpretation of the experiment's results, software engineering deals with differing contexts that affect the input and the results. In software engineering the technologies are mostly human intensive rather than automated. Like manufacturing the major problem is understanding and improving the relationship between processes and the products they create. But unlike manufacturing, the process in software engineering is development not production. So we cannot collect data from exact repetitions of the same process. We have to build our models at a higher level of abstraction but still take care to identify the context variables.

Currently, there is an insufficient set of models that allow us to reason about the discipline, a lack of recognition of the limits of technologies for certain contexts, and insufficient analysis and experimentation going on but this latter situation is improving as evidenced by this textbook.

This book is a landmark in allowing us to train both the researcher and practitioner in software engineering experimentation. It is a major contribution to the field. The authors have accumulated an incredible collection of knowledge and packaged it in an excellent way, providing a process for scoping, planning, running, analyzing and interpreting, and packaging experiments. They cover all necessary topics from threats to validity to statistical procedures.

It is well written and covers a wide range of information necessary for performing experiments in software engineering. When I began doing experiments, I had to find various sources of information, almost always from other disciplines, and adapt them to my needs as best I could. If I had this book to help me, it would have saved me an enormous amount of time and effort and my experiments would probably have been better.

January 2012 Professor Victor R. Basili

Foreword from the First Edition

I am honored to be asked to write a foreword for this revision of the authors' book with the same title that was published in 2000. I have used the original edition since its publication as a teacher and a researcher. Students in my courses at Colorado State University, Washington State University, University of Denver, and Universitaet Wuerzburg have used the book over the years. Some were full-time employees at major companies working on graduate degrees in Systems Engineering, others full-time Masters and Ph.D. students. The book worked well for them. Besides the treatment of experimental software engineering methods, they liked its conciseness. I am delighted to see that the revised version is as compact and easy to work with as the first.

The additions and modifications in this revised version very nicely reflect the maturation of the field of empirical software engineering since the book was originally published: the increased importance of replication and synthesis of experiments, and the need of academics and professionals to successfully transfer new technology based on convincing quantitative evidence. Another important improvement concerns the expanded treatment of ethical issues in software engineering experimentation. Especially since no formal code of ethics exists in this field, it is vitally important that students are made aware of such issues and have access to guidelines how to deal with them.

The original edition of this book emphasized experiments. In industry, however, case studies tend to be more common to evaluate technology, software engineering processes, or artifacts. Hence the addition of a chapter on case studies is much needed and welcomed. So is the chapter on systematic literature reviews.

Having taught a popular quantitative software engineering course with the original edition for a dozen years, this revised version with its additions and updates provides many of the materials I have added separately over the years. Even better, it does so without losing the original edition's compactness and conciseness. I, for one, am thrilled with this revised version and will continue to use it as a text in my courses and a resource for my student researchers.

January 2012 Professor Anneliese Amschler Andrews

Preface

Have you ever had a need to evaluate software engineering methods or techniques against each other? This book presents experimentation as one way of evaluating new methods and techniques in software engineering. Experiments are valuable tools for all software engineers who are involved in evaluating and choosing between different methods, techniques, languages and tools, or solutions.

It may be that you are a software practitioner, who wants to evaluate methods and techniques before introducing them into your organization. Alternatively, you want to assess how users experience different possible solutions, whether being two Web page designs or different versions of a feature. You may also be a researcher, who wants to evaluate new research results against something existing, to get a scientific foundation for your new ideas. You may be a teacher, who believes that knowledge of empirical studies in software engineering is essential to your students. Finally, you may be a student in software engineering who wants to learn some methods to turn software engineering into a more scientific discipline and to obtain quantitative data when comparing different methods and techniques. This book provides guidelines and examples of how you should proceed to succeed in your mission.

Software Engineering and Science

The term "software engineering" was coined in 1968, and the area continues to mature. Software engineering has for a long time been driven by technology development and advocacy research, the latter referring to the fact that we have invented and introduced new methods and techniques over the years based on marketing and conviction rather than scientific results. The situation has improved over time. To some extent, this evolution is understandable with the pace at which the information society has established itself during the last couple of decades. However, we need to continue improving research in software engineering if we want to have control of the software we develop. Control comes from being able

to evaluate new methods, techniques, languages, and tools before using them. Moreover, this would help us turn software engineering into a more scientific discipline. Before looking at the issues we must address to turn software engineering into science, let us look at the way science is viewed in other areas.

In "Fermat's Last Theorem" by Dr. Simon Singh [232], science is discussed. The essence of the discussion can be summarized as follows. In science, physical phenomena are explored by putting forward hypotheses. The phenomenon is observed and if the observations are in line with the hypothesis, this becomes evidence for the hypothesis. The intention is also that the hypothesis should enable prediction of other phenomena. Experiments are important to test the hypothesis and in particular the predictive ability of the hypothesis. If new experiments support the hypothesis, then we have more evidence in favor of the hypothesis. As the evidence grows and becomes strong, the hypothesis can be accepted as a scientific theory.

In summary, scientific advancement is driven by hypothesis testing through empirical research. This may not be the way most research is conducted in software engineering today. However, the need to evaluate and validate new research proposals by conducting empirical studies is acknowledged to a higher degree today than before. Empirical studies include, for example, systematic literature studies, surveys, experiments, and case studies. Thus, the objective of this book is to introduce and promote the use of empirical studies in software engineering with a particular emphasis on experimentation.

Purpose

The purpose of the book is to introduce students, teachers, researchers, and practitioners to experimentation and empirical evaluation with a focus on software engineering. The objective is in particular to provide guidelines on how to perform experiments to evaluate methods, techniques, and tools in software engineering. To do this a decision-making structure for selecting an appropriate research design is presented. Furthermore, short introductions are also provided for other empirical approaches than experiments. The introduction to experimentation is provided through a process perspective. The focus is on the steps that we have to go through to perform an experiment. The process can be generalized to other types of empirical studies, but the main focus here is on experiments and quasi-experiments.

The motivation for the book comes from the need for support we experienced when making our software engineering research more empirical. Several books are available which either treat the subject in very general terms or focus on some specific part of experimentation, most of them focusing on the statistical methods in experimentation. These are important, but there is a lack of books elaborating on experimentation from a process perspective. Moreover, there are few books addressing experimentation in software engineering in particular, and actually there was no book at all when the original edition of this book was published. Apparently the book filled a gap, as it is cited more than 10,000 times, according to Google

Scholar, and we now offer a revised version to also include the recent advancements in the field.

Scope

The scope of the book is primarily experiments in software engineering as a means for evaluating methods, techniques, etc. However, the book also provides information on empirical research in general and some further details concerning systematic literature reviews, surveys, and case studies. The intention is to provide a brief understanding of these research approaches and in particular to relate them to experimentation.

The chapters of the book cover different steps to go through to perform experiments in software engineering. Moreover, examples of empirical studies related to software engineering are provided throughout the book. It is of particular importance to illustrate for software engineers that empirical studies and experimentation can be practiced successfully in software engineering. Two examples of experiments are included in the book. The examples may not be perceived as contemporary. However, the technical areas of the experiments are not essential. The objective of the examples is to illustrate the experiment process and to exemplify how software engineering experiments can be performed and reported. The intention is that these studies should work as good examples and sources of inspiration for further empirical work in more contemporary areas of software engineering.

The book is mainly focused on experiments, but it should be remembered that other research approaches are also available, as discussed in Chap. 2.

Target Audience

The target audience of the book can be divided into four categories.

Students may use the book as an introduction to experimentation in software engineering with a particular focus on evaluation. The book is suitable as a course book in undergraduate or graduate studies where the need for empirical studies in software engineering is stressed. Exercises and project assignments are included in the book to combine the more theoretical material with some practical aspects.

Teachers may use the book in their classes if they believe in the need for making software engineering more empirical. The book is suitable as an introduction to the area. It should be fairly self-contained, although an introductory course in statistics is recommended.

Researchers may use the book to learn more about how to conduct empirical studies and use them as one important ingredient in their research. Moreover, the objective is that it should be fruitful to come back to the book and use it as a checklist when performing empirical research.

Practitioners may use the book for conducting A/B testing, i.e., to evaluate alternative solutions concerning, for example, two different Web page solutions or two different implementations of a feature. Furthermore, the book may be used as a "cookbook" when evaluating some new methods or techniques before introducing them into their organization. Practitioners are expected to learn how to use empirical studies in their daily work when evaluating solutions or changing, for example, the development process in the organization in which they are working.

Outline

The book is divided into three parts. The outline of the book is summarized in Tables 1 and 2, which also show a mapping to the previous edition of this book, published in 2012. In general, the book has been updated to capture the evolution of empirical studies in software engineering with a particular focus on experiments.

Part I provides a general introduction to the area of empirical studies in Chap. 1. Chapter 2 puts empirical studies in general and experiments in particular into a software engineering context through a decision-making structure for selecting an appropriate research design for the research question posed. In Chap. 3, several areas that are essential in empirical research are presented. The chapter discusses ethics in empirical studies, replications, aggregation of evidence, theory in a software engineering context, measurement, empirical studies for improving software development, and empirically based technology transfer. In Chap. 4, we provide an overview of how to conduct systematic literature reviews, and to synthesize findings from several empirical studies. Next, we introduce surveys, experiments, and case studies. An introduction to survey research is presented in Chap. 5. Chapter 6 provides a general introduction to experimentation by introducing an experiment process. Chapter 7 gives an overview of case study research as a related type of empirical study.

Part II has one chapter for each step in the experiment process. Chapter 8 discusses how to set the scope for an experiment and Chap. 9 focuses on the planning step. Operation of the experiment is discussed in Chaps. 10 and 11 present methods for analyzing and interpreting the results. Chapter 12 discusses presentation and packaging of the experiment.

Part III contains two example experiments. In Chap. 13, an example is presented where the main objective is to illustrate the experiment process, and the example in Chap. 14 is used to illustrate how an experiment in software engineering may be reported in a research paper.

Some exercises and data are presented in Appendix A. Finally, the book displays some statistical tables in Appendix B. The tables are primarily included to provide support for some of the examples in the book. More comprehensive tables are available in most statistics books.

Table 1 Structure of Part I of the book

Chapter title	Revised version	Previous edition (2012)	Main updates
Introduction	1	1	
Empirical Research	2		New chapter. A decision-making structure for selecting a research design is presented. Two sections from Chapter 2 in the previous edition are included
Essential Areas in Empirical Research	3		New chapter including new sections as well as sections from Chapter 2 in the previous edition. Furthermore, the previous measurement chapter has been moved to this chapter. Moreover, information concerning open science has been added to the chapter
Systematic Literature Reviews	4	4	The chapter is a consolidation based on information provided in different places in the previous version of the book. Furthermore, the chapter has been updated with additional descriptions, for example, concerning the need for updating systematic literature studies. The objective has been to present a more comprehensive description of systematic literature reviews
Surveys	5		New chapter. The chapter describes survey research
Experiments	6	6	The chapter is a consolidation based on information provided in different places in the previous version of the book. Moreover, the chapter has been updated with additional descriptions, including A/B testing. The objective has been to present a more comprehensive description of experimentation
Case Studies	7	5	The chapter is a consolidation based on information provided in different places in the previous version of the book. Furthermore, the chapter has been updated with additional descriptions. The objective has been to present a more comprehensive description of case study research

Exercises

The exercises are divided into four categories; the first category is presented at the end of each chapter in Parts I and II of the book, i.e., Chaps. 1–12, and the other three in Appendix A:

Table 2 Structure of Parts II and III as well as the Appendices of the book

Chapter title	Revised version	Previous edition (2012)	Main updates
Part II. Steps in the Experiment Process			
Scoping	8	7	
Planning	9	8	The chapter has been updated with additional information concerning paired designs and validity evaluation and threats
Operation	10	9	
Analysis and Interpretation	11	10	A new subsection has been added related to the use of tools for statistical analysis
Presentation and Package	12	11	
Part III. Example Experiments			
Experiment Process Illustration	13	12	
Are the Perspectives Really Different?	14	13	
Appendices			
Exercises	A	A	
Statistical Tables	B	B	

Understanding Five questions capturing the most essential points are provided at the end of each chapter. The objective is to ensure that the reader has understood the most important concepts.

Training These exercises provide an opportunity to practice experimentation. The exercises are particularly targeted toward analyzing data and answering questions in relation to an experiment. The data in an Excel file and the answers to the training exercises can be found through the following link: https://portal.research.lu.se/en/publications/experimentation-in-software-engineering-2024-edition.

Reviewing This exercise is aimed at the examples of experiments presented in Chaps. 13–14. The objective is to give an opportunity to review some presented experiments. After having read several experiments presented in the literature, it is clear that many experiments suffer from certain problems. This is mostly due to the inherent problems of performing experimentation in software engineering. Instead of promoting criticism of work by others, we have provided some examples of studies that we have conducted ourselves. They are, in our opinion, representative of the type of experiments that are published in the literature. This includes the fact that they have their strengths and weaknesses.

Assignments The objective of these exercises is to illustrate how experiments can be used in evaluation. These assignments are examples of studies that can be carried out within a course, either at a university or in industry. They are deliberately aimed at problems that can be addressed by fairly simple experiments. The assignments can be done after reading the book or one of the assignments can be carried out as the book is read. The latter provides an opportunity to practice while reading the chapters. As an alternative, we would like to recommend that teachers formulate an assignment, within their area of expertise, that can be used throughout the book to exemplify the concepts presented in each chapter.

Karlskrona, Sweden — Claes Wohlin
Lund, Sweden — Per Runeson
Malmö, Sweden — Martin Höst
Malmö, Sweden — Magnus C. Ohlsson
Lund, Sweden — Björn Regnell
Lund, Sweden — Anders Wesslén
March 2024

Acknowledgments

It is a sincere pleasure to introduce the third edition of the book. We do appreciate all positive feedback received since the publication of the first edition 24 years ago. This edition is based on *Experimentation in Software Engineering: An Introduction*, published in 2000, and the revised edition titled *Experimentation in Software Engineering* published in 2012. Moreover, the book was published in Chinese in 2015. This new book edition is both a revision and an extension of the former editions.

A book is rarely just an achievement by the authors. Support and help from several persons, including families, friends, colleagues, international researchers, and funding organizations, are often prerequisites for a book. This book is certainly no exception. In particular, we would like to express our sincere gratitude to the readers of the two previous editions of the book. Your use of the book has been a great source of inspiration and a motivation to publish the current version. In particular, we would like to thank Prof. Alan Kelon Oliveira de Moraes, Brazil, for sending the email that triggered the publication of the book's second edition. Moreover, we are grateful to Ralf Gerstner, Executive Editor at Springer, for his help and support with the book over the years, particularly for suggesting that we publish a third edition. Furthermore, we thank the following individuals for contributing to the book.

First, we would like to thank the first primary external user of the book's first edition, Prof. Giuseppe Visaggio, for adopting a draft of this book in one of his courses and for providing valuable feedback. We want to express our gratitude to Prof. Anneliese Andrews and Prof. Khaled El Emam for encouraging us to publish the book in the first place and for valuable comments. We also would like to thank Prof. Lionel Briand, Prof. Christian Bunse, and Dr. John Daly for providing the data for the example of object-oriented design. We also thank Dr. Thomas Thelin for allowing us to include an experiment he did with two of the book's authors.

Early book drafts were used and evaluated internally within the Software Engineering Research Group at Lund University. Thus, we would like to thank the group members for providing feedback on different drafts of the book. In particular, we would like to thank Dr. Lars Bratthall for thoroughly reviewing the manuscript

and providing valuable comments. We also want to acknowledge the anonymous reviewers' contributions to the book.

For the second edition of the book, we received valuable input and improvement proposals, and hence, we would like to thank the following individuals for their valuable input: Prof. Anneliese Andrews, Prof. David Budgen, Prof. Jürgen Börstler, Prof. Samuel Fricker, Prof. Barbara Kitchenham, Prof. Dieter Rombach, and Prof. Richard Torkar. Thanks also to Jesper Runeson for work on the LaTeX transformation of the book.

Moreover, we sincerely thank Prof. Qing Wang and her colleagues, who initiated the translation of the book into Chinese. During the translation process, some questions arose that have contributed to further improving the new edition.

In addition to the above individuals, we would also like to thank all members of ISERN (International Software Engineering Research Network) for the exciting and enlightening discussion regarding empirical software engineering research over the years.

For the chapter on case studies, we are grateful for the feedback on the checklists from the ISERN members and attendees of the International Advanced School of Empirical Software Engineering in September 2007. A special thanks to Dr. Kim Weyns and Dr. Andreas Jedlitschka for their review of an early draft of the chapter.

For the third edition, we sincerely thank Dr. Aybüke Aurum for allowing us to use the decision-making structure presented in a joint article with Prof. Claes Wohlin. The structure supports the selection of an appropriate research design. Furthermore, it puts different research methodologies and methods into a broader context. Moreover, we are grateful to Dr. Johan Linåker, Dr. Sardar Muhammad Sulaman, and Dr. Rafael Maiani de Mello for letting us use their technical report on conducting survey research in software engineering as a source of inspiration for the survey chapter in the book. The technical report was co-authored with Prof. Martin Höst.

Numerous research projects at Lund University and Blekinge Institute of Technology have contributed to the book over the years. Different grants have funded research projects where empirical studies have been a cornerstone and helped shape the experience we have tried to document through the book. This book is a result of all these research projects.

Karlskrona, Sweden — Claes Wohlin
Lund, Sweden — Per Runeson
Malmö, Sweden — Martin Höst
Malmö, Sweden — Magnus C. Ohlsson
Lund, Sweden — Björn Regnell
Lund, Sweden — Anders Wesslén
March 2024

Contents

Part I
Context

Chapter 1
Introduction

The information technology revolution has meant, among other things, that software has become a part of more and more products. Software is found in products ranging from toys to space shuttles. This means that a vast amount of software has been and is being developed. Software development is by no means easy; it is a highly creative process. The rapid growth of the area has also meant that numerous software projects have run into problems in terms of missing functionality, cost overruns, missed deadlines, and poor quality. These problems or challenges were identified already in the 1960s, and in 1968 the term "software engineering" was coined with the intention of creating an engineering discipline that focused on the development of software-intensive systems.

Software engineering is formally defined by the Institute of Electrical and Electronics Engineers (IEEE) [115] as follows: "*The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*" Software engineering in general is presented and discussed in books by, for example, Sommerville [238], van Vliet [258], and Pfleeger and Atlee [195]. The objective here is to present how empirical studies and experiments in particular fit into a software engineering context. Three aspects in the definition above are of particular importance here. First, it implies a structure for developing software through pointing at different life cycle phases; second, it stresses the need for a systematic and disciplined approach; finally, it highlights the importance of quantification. The use of empirical studies is related to all three of them. The software engineering context is further discussed in Sect. 1.1. The need to make software engineering more scientific and how empirical studies play an important role in this is discussed in Sect. 1.2.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_1

1.1 Software Engineering Context

A software process model is an abstract representation of the phases to go through and the activities to perform when developing software. Examples of software process models are the waterfall model, incremental development, evolutionary development, the spiral model, and different agile and continuous approaches to software development. These and other models are discussed in the general software engineering literature. A simplistic view of the software process is shown in Fig. 1.1. It should be noted that the process is crucial whether we work with development of a new product, evolution of an existing product, or maintenance of a product.

In Fig. 1.1, an idea and resources, primarily in the form of people, are inputs to the software process, and the people develop a software product going through the different phases and performing the different activities in the software process.

The development of software products is often a complex task. Software projects may run over a long period of time and involve many people (even if using agile methods), due to the complexity of the software products that are developed. This implies that the software process often also becomes very complex. It consists of many different activities and many artifacts are written before the final product can be delivered. The complexity of the software process means that it is difficult to optimize it or even find a good enough process. Thus, it is important for companies to strive to improve their way of doing business if they intend to stay competitive. This means that most companies are continuously trying to improve the way they develop software to improve the products, lower the cost, and so forth. The software process stresses the need for a systematic and disciplined approach to working. Being agile is not an exception; there is still a need to have a structured approach, although agile methods stress the need to not document too much and emphasize the need to have running code continuously instead of "only" at the end of a large project. The requirement from tool chain and decision structures for continuous integration and deployment (CI/CD) clearly indicates this need for a structured approach. A systematic and disciplined approach is also needed when improving the software process, and hence a way to improve process is needed.

An example of an improvement process tailored for software development is the Quality Improvement Paradigm (QIP), defined by Basili [16]. It consists of several activities to support a systematic and disciplined approach to improvement. A more general improvement process is the well-known Plan/Do/Study/Act cycle [33, 57].

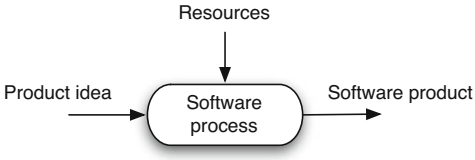


Fig. 1.1 An illustration of the software process

The improvement processes include two activities that we would like to highlight, although the same terminology is not always used:

- Assessment of the software process
- Evaluation of a software process improvement proposal

The assessment is conducted to identify suitable areas for improvement. Assuming that it is possible to identify areas for improvement through some form of assessment, the next step is to determine how these areas of improvement may be addressed to cope with the identified problems. For example, if too many defects are found in system testing, it may be possible to improve earlier testing, reviews, or even specific parts in the development, for example, software design. The objective is that the assessment of the current situation and knowledge about the state-of-the-art should result in concrete process improvement proposals being identified. When the improvement proposals have been identified, it is necessary to determine which to introduce, if any. It is often not possible just to change the existing software process without having more information about the actual effect of the improvement proposal. In other words, it is necessary to evaluate the proposals before making any major changes.

One problem that arises is that a process improvement proposal is very hard to evaluate without direct human involvement. For a product, it is possible to first build a prototype to evaluate whether it is something to work further with. For a process, it is not possible to build a prototype. It is possible to make simulations and compare different processes, but it should be remembered that this is still an evaluation that is based on a model. The only real evaluation of a process or process improvement proposal is to have people using it, since the process is just a description until it is used by people. Empirical studies are crucial to the evaluation of processes and human-based activities. It is also beneficial to use empirical studies when there is a need to evaluate the use of software products or tools. Experimentation provides a systematic, disciplined, quantifiable, and controlled way of evaluating human-based activities. This is one of the main reasons why empirical research is common in social and behavioral sciences (see, e.g., Robson [207]).

In addition, empirical studies and experiments in particular are also important for researchers in software engineering. New methods, techniques, languages, and tools should not just be suggested, published, and marketed. It is crucial to evaluate new inventions and proposals in comparison with existing ones. Experimentation provides this opportunity, and should be used accordingly. In other words, we should use the methods and methodologies available when conducting research in software engineering. This is further discussed in Chap. 2.

1.2 Science and Software Engineering

Software engineering is a cross-disciplinary subject. It stretches from technical issues such as databases and operating systems, through language issues, for example, syntax and semantics, to social issues and psychology. Software development

is human-intensive; we are, so far, unable to manufacture new software, although advanced AI may increase such opportunities. It is a discipline based on creativity and the ingenuity of the people working in the field. Nevertheless, we should, when studying and doing research in software engineering, aim at treating it as a scientific discipline. This implies using scientific methods for doing research and when making decisions regarding changes in the way we develop software.

In order to perform scientific research in software engineering, we have to understand the methods that are available to us, their limitations, and when they can be applied. Software engineering stems from the technical community. Thus, it is natural to look at the methods used for research in, for example, hardware design and coding theory, but based on the nature of software engineering we should look at other disciplines too [170]. Glass summarized four general research approaches in the field of software engineering [88]. They were initially presented in a software engineering context by Basili [18]. The general approaches are:

Scientific	The world is observed and a model is built based on the observation, for example, a simulation model.
Engineering	The current solutions are studied and changes are proposed, and then evaluated.
Empirical	A model is proposed and evaluated through empirical studies, for example, case studies or experiments.
Analytical	A formal theory is proposed and then compared with empirical observations.

The engineering method and the empirical method can be seen as variations of the scientific method [18].

Traditionally, the analytical method is used in the more formal areas of electrical engineering and computer science, e.g., electromagnetic theory and algorithms. The scientific method is used in applied areas, such as simulating a telecommunication network to evaluate its performance. It should, however, be noted that simulation as such is not only applied in the scientific method. Simulation may be used as a means for conducting an experiment as well. The engineering method is probably dominating in industry.

The empirical studies have traditionally been used in social sciences and psychology, where we are unable to state any laws of nature, as in physics.[1] The fields of social sciences and psychology are concerned with human behavior. The important observation, in this context, is hence that software engineering is very much governed by human behavior through the people developing software. Thus, we cannot expect to find any formal rules or laws in software engineering except perhaps when focusing on specific technical aspects. The focus of this book is on applying and using empirical methods in software engineering. The objective is in particular to emphasize the underlying process when performing empirical studies

[1] Lehman [156] referred to laws of software evolution, but this notion has not been widespread in subsequent work on theory. The concept of theory in software engineering is further discussed in Sect. 3.3.

in general and experimentation in particular. An experiment process is presented which highlights the basic steps to perform experiments, guidance is provided on what to do, and the steps are illustrated using software engineering examples.

It must be noted that it is not claimed that the analytical, scientific, and engineering methods are inappropriate for software engineering. They are necessary for software engineering as well; for example, we may build mathematical models for software reliability growth [163]. Moreover, the research methods are not orthogonal, and hence it may, for example, be appropriate to conduct an empirical study within the engineering method. The important point is that we should make better use of the methods available within empirical research. They are frequently used in other disciplines, for example, behavioral sciences, and the nature of software engineering has much in common with disciplines outside the technical parts of engineering.

The very first experiments in software engineering were conducted in the late 1960s by Grant and Sackmann [95] about on- and offline work in testing, according to Zendler [285]. In the 1970s, a few pioneers conducted experiments on structured programming [162], flowcharts [222], and software testing [184]. The need for systematic experimentation in software engineering was emphasized in the mid-1980s by Basili et al. [24]. Other articles stressing the need for empiricism in software engineering have since been published (see, e.g., work by Basili [18], Fenton [77], Glass [88], Kitchenham et al. [136], Potts [202], and Tichy [251]). The lack of empirical evidence in software engineering research has over the years been stressed by, for example, Tichy et al. [252], Zelkowitz and Wallace [284], and Glass et al. [89]. The situation has improved since these publications, but a more scientific approach to software engineering is still needed. The focus of this book is on software engineering and the application and use of empirical studies, in particular experimentation, in software engineering.

Empirical research approaches in software engineering include, but are not limited to:

- Conducting systematic literature studies
- Performing surveys through, for example, questionnaires
- Setting up formal experiments
- Studying real projects in industry, for example, performing a case study

These approaches and some others are described in Chap. 2 in the context of a decision-making structure for selecting an appropriate research approach. The four approaches above are then discussed in some more detail in Chaps. 4–7 before focusing the remainder of this book on experimentation. A more general introduction to other research approaches than experiments is presented by, for example, Robson [207]. Case studies in general are elaborated by Yin [281] and case studies specifically in software engineering are elaborated by Runeson et al. [212]. The research approaches are neither completely orthogonal nor competing. They provide a convenient classification, but some studies may be viewed as combinations of them or somewhere between two of them. Thus, there are both similarities and differences between the research approaches.

The main reason to use experimentation in software engineering is to enable understanding and identification of relationships between different factors, or variables. A number of preconceived ideas exist, but are they true? Does object-orientation improve reuse? Are reviews cost-effective? Should we have review meetings or is it sufficient to hand in the remarks to a moderator? These types of questions can be investigated to improve our understanding of software engineering. An improved understanding is the basis for changing and improving the way we work; hence empirical studies in general and experimentation in particular are essential.

The introduction to experimentation in software engineering is done by presenting a process for experimentation. The basic steps in the process can be used for other types of empirical studies too. The focus is, however, on providing guidelines and support for performing experiments in software engineering. Furthermore, it should be noted that "true" experiments, i.e., experiments with full randomization, are difficult to perform in software engineering. Software engineering experiments are often quasi-experiments, i.e., experiments in which, for example, it has not been possible to assign participants in the experiments to groups at random [50]. Quasi-experiments are important, and they can provide valuable results. The process presented in this book is aimed at both "true" experiments and quasi-experiments, including A/B testing. Quasi-experiments are particularly supported by a thorough discussion of threats to validity in software engineering experimentation.

Thus, the intention of this book is to provide an introduction to empirical studies and particularly to experimentation, to highlight the opportunities and benefits of doing experiments in the field of software engineering. The empirical research method can, and should, be used more in software engineering. Some arguments against empirical studies in software engineering are refuted by Tichy et al. [251]. It is hoped that this practical guide to experimentation in software engineering facilitates the use of empirical studies and experimentation both within software engineering research and practice.

1.3 Exercises

1.1 What aspects of the IEEE definition of software engineering are most important, and how does it relate to empirical studies?

1.2 Why are empirical studies essential in software engineering?

1.3 Why can experiments be viewed as prototyping for process changes?

1.4 How can experiments be used in improvement activities?

1.5 When is the empirical research method best suited in software engineering in comparison with the scientific, engineering, and analytic methods respectively?

Chapter 2
Empirical Research

The overall objective of this chapter is to introduce empirical research. More specifically, the objectives are: (1) to introduce and discuss a decision-making structure for selecting an appropriate research approach, (2) to compare a selection of the introduced research methodologies and methods, and (3) to discuss how different research methodologies and research methods can be used in different stages in a research project or when pursuing, for example, PhD studies. To fulfill the first objective, an overview of the decision-making structure for selecting a specific research methodology or method is provided in Sects. 2.1–2.4. The decision-making structure is adapted from Wohlin and Aurum [270]. The second objective is discussed in Sect. 2.6 and the third objective is elaborated in Sect. 2.7.

2.1 Selecting a Research Design

The book is focused on experimentation. However, experimentation exists in a context of multiple approaches to empirical research. When performing research, it is crucial to select an appropriate research approach for the planned research. Thus, we introduce a decision-making structure in Fig. 2.1 for selecting a research design. In the experimentation context, the structure functions as a tool for researchers to decide to perform an experiment when appropriate for their research. The decision-making structure includes eight areas, or decision points, in empirical research design, which is divided into three phases used in business management [7], and adapted to research design and discussed in more detail by Wohlin and Aurum [270]. The structure is intended to support researchers in defining an appropriate research design for their research. All decision points should be gone through before conducting any research study. The outcome from the decision points is a research design which is then used to conduct the research study and obtain the research

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_2

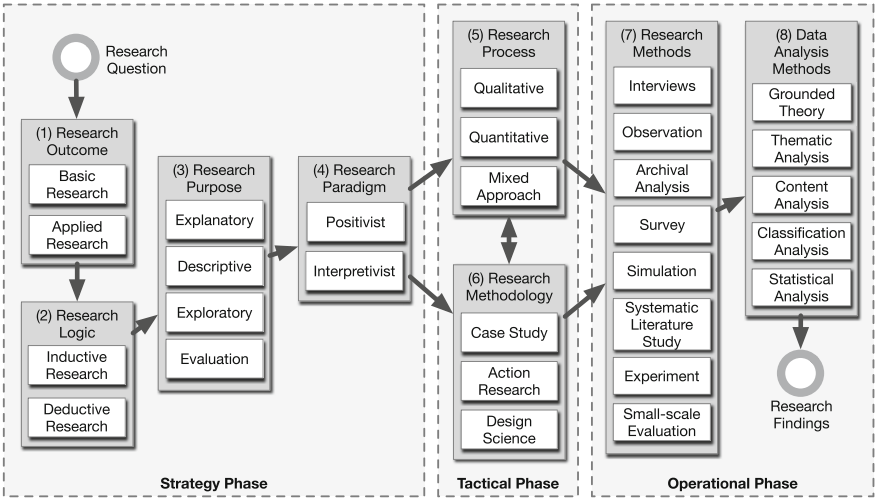


Fig. 2.1 Decision-making structure to select research design. (The figure is adapted from [270]. The copyright for the original figure is held by Springer Science+Business Media New York 2014)

findings as shown in Fig. 2.1. The three phases, which are further elaborated below, are as follows:

1. The *Strategy phase* is intended to set the basis for the overall research, and ensure that the researchers perform the research systematically. The phase includes making decisions related to research question, research logic, research purpose, and research paradigm.
2. The *Tactical phase* is concerned with decisions on how to conduct the research to address the overall research question. In this phase, decisions related to the research process and potentially framing the research in a methodology are made.
3. The *Operational phase* is related to decisions on how to implement the research. The decisions are focused on the research method, particularly about data collection, and data analysis techniques. The phase ends with formulating the research findings in relation to the research question posed in the strategy phase.

The research approach can be a research methodology or research method, or both. Here, we distinguish research methodology and research method as follows. A *research methodology* is a combination of research methods, processes, and frameworks. This implies that research methodologies do not have any prescribed data collection method as such or a specific objective, for example, in terms of observation, evaluation, or experimentation. Thus, a research methodology can be particularly suitable when more than one data collection method is needed in the research. For example, when conducting research studies in an industrial setting, more than one data collection method is often needed. There can be a need for both interviewing people and going through some types of artifacts. Selection of an

appropriate research methodology in industry–academia collaboration is discussed by Wohlin and Runeson [272]. *Research methods* are more directly focused on how the data should be collected and for what objective. However, the borderline between what constitutes a research methodology and a research method is fuzzy, which also means that there are inconsistent views among researchers and hence also in publications. When discussing research methodologies or research methods in general terms, we refer to them as jointly being research approaches.

The three phases in the decision-making structure are presented in the following sections.

2.2 Strategy Phase

The strategy phase includes four decisions points (Decision points 1–4) as shown in Fig. 2.1. The research question is the starting point of a research study, or more specifically here an empirical research study. The research question should address a significant and relevant problem. Furthermore, the research question strongly influences the decisions made during the conduct of a research study. As more is learned through the research, the research question may evolve, typically becoming more specific as the research progresses.

Decision point 1 relates to the expected research outcome. According to Collis and Hussey [49], research may be classified as basic or applied. *Basic research* relates to an increased understanding where the objective is to generate knowledge. *Applied research* focuses on finding a solution to a practical problem or an improvement of some existing practice. Applied research builds on the knowledge from basic research.

Next, the research logic needs to be addressed, i.e., Decision point 2. Research logic is concerned with the direction of the research, i.e., the research may evolve from being general to being more specific or vice versa [49]. Two common ways of reasoning in research are deductive and inductive research [52].

Deductive research evolves from the general to the more specific. For example, a researcher may start with a hypothesis, based on existing knowledge, potentially expressed as a theory. Then, the researcher wants to evaluate the hypothesis based on the data collected. It is a top-down approach and it is often referred to as theory-testing research. *Deductive research* is often quantitative since the researcher often wants to test a theory using empirical data. On the other hand, *inductive research* evolves from the specific to the more general. It is based on inductive arguments, and it is more qualitative, although it may include also collecting quantitative data. The researcher makes observations from the data with the aim to detect patterns, which then can be used to generate general conclusions or theories. Inductive research is a bottom-up approach and often referred to as being theory-building research.

Decision point 3 relates to the purpose of the research. Collis and Hussey [49] classify the purpose of the research into four potentially different purposes: exploratory, descriptive, explanatory, and evaluation research.

Exploratory research focuses on studying objects in their natural setting and letting the findings emerge from the observations. This implies that a *flexible research design* [5] is needed to adapt to changes in the observed phenomenon. Flexible design research is also referred to as *qualitative research*, as it is primarily informed by qualitative data. Inductive research attempts to interpret a phenomenon based on explanations that people bring forward. It is concerned with discovering causes noticed by the subjects in the study, and understanding their view of the problem at hand. The subject is the person who is taking part in an empirical study to evaluate an object.

Descriptive research is used to, for example, describe a phenomenon or characteristics of a problem. The research question in descriptive research typically starts with "What" or "How," since the intention is to describe [49] an existing phenomenon. For example, descriptive research may be used to describe how risk management is conducted in agile projects.

Explanatory research focuses mainly on quantifying a relationship or on comparing two or more groups with the aim to identify a cause–effect relationship. The research is often conducted through setting up a controlled experiment. This type of study is a *fixed design* [5] study, implying that factors are fixed before the study is launched. Fixed design research is also referred to as *quantitative research*, as it primarily is informed by quantitative data. Quantitative investigations[1] are appropriate when testing the effect of some manipulation or activity. An advantage is that quantitative data promotes comparisons and statistical analyses. It is possible for qualitative and quantitative research to investigate the same topics but each of them will address a different type of research question. For example, a quantitative investigation could be launched to investigate how much a new inspection[2] method decreases the number of faults found in a test. To answer questions about the sources of variations between different inspection groups, a qualitative investigation could be launched.

Evaluation research is used to evaluate the value of, for example, different techniques, methods, processes, or tools. For example, we may want to evaluate pair programming versus a single programmer in terms of effectiveness, efficiency, and software quality. Runeson and Höst [210] call this type of research "improving" in an engineering context. Here, the classification by Collis and Hussey [49] is used. Evaluation research is a somewhat broader concept since it includes all types of evaluation. In empirical software engineering, evaluation of different techniques, methods, processes, and tools is common.

Fixed design strategies, such as controlled experiments, are appropriate when testing the effects of a treatment, while a flexible design study of beliefs, understandings, and multiple perspectives is appropriate to find out why the results are as

[1] The term "investigation" is used as a more general term than a specific study.

[2] It is sometimes also referred to as a review or a code review, if reviewing code. However, we have chosen to use the term "inspection" to avoid mixing it up with a systematic literature review.

they are. The four different research purposes should be regarded as complementary rather than competitive.

Decision point 4, the final decision point in the strategy phase, is concerned with the research paradigm. Easterbrook et al. [65] point out that underlying research philosophies are seldom expressed explicitly in software engineering research. They present the following four paradigms: positivism, constructivism (or interpretivism, or alternatively interpretive), critical research, and pragmatism. Here the focus is on the following two paradigms: positivist and interpretivist. These two paradigms may be exemplified with the following studies in software engineering: The positivist paradigm is used by Staron et al. [243] when conducting experiments on UML models, and the interpretivist paradigm is applied by Moe et al. [178] when performing a case study in industry.

In *positivist research*, it is assumed that the researcher and the reality are separated. The paradigm advocates objective research. The research is believed to be objective and reliable if it is replicable by others and they can be expected to reach similar conclusions [148]. Replication and other essential areas in empirical software engineering are further discussed in Chap. 3. The intention in quantitative research is often to study cause and effect and being able to state general conclusions. Quantitative methods are primarily used. The positivist paradigm has a tendency to apply an explanatory or evaluation research approach. Different research methods may be applied, and some of the most common methods are controlled experiment, survey, and archival analysis.

Interpretivist research is focused on understanding human activities in a specific context. The research is conducted from the participants' perspective [148]. The aim of interpretivist research is to obtain a deep understanding of a phenomenon. The phenomenon is studied within its cultural and contextual situation, i.e., it is studied in its natural setting from the viewpoint of the participants. It is assumed that the validity of the research is obtained by collecting qualitative data, for example, using research methodologies such as case study and action research, often including interviews or observations for data collection.

2.3 Tactical Phase

The tactical phase includes two decision points: research process and research methodology. The researcher may enter the tactical phase either by going to Decision point 5 or 6. Depending on the needs, the researcher may either address both decision points or address one of them before moving to the operational phase.

Decision point 5 is focused on the research process, or more specifically the type of data to be collected. There are two main directions, i.e., the collection of qualitative or quantitative data. A third alternative exists, which is called a mixed approach [52]. The latter includes collection of both qualitative and quantitative data. The mixed approach is often related to a research methodology, such as case

study research or action research, which both include using more than one research method for data collection.

Qualitative research is based on rich descriptions in natural language or other informal representations. The data collection is often done by using research methods such as interviews, archival analysis, or observations. The objective is to understand and explain social phenomena, i.e., in the context of the research subjects. Klein and Myers [148] argue that qualitative research is not a synonym for interpretivist research, although most qualitative research is related to interpretivist research. Their viewpoint is that qualitative research can be either interpretivist or positivist research.

Quantitative research is concerned with either collecting quantitative data directly or using qualitative data that has been quantified. This includes identifying causal relationships. In quantitative research, the focus is on measurement and statistical analysis. Measurement is described in further detail in Chap. 3, while statistical analysis for experimentation is presented in Chap. 11. Quantitative research can be exemplified with research methods such as controlled experiments, surveys through questionnaires, and simulation.

Given the complexity of research in software development, more than one research methodology or research method is needed. For example, different research methodologies or research methods are needed in a research project or when pursuing PhD studies. This aspect is further discussed in Sect. 2.7.

After having finalized Decision point 5, it is possible to either move to Decision point 6 (research methodology) or to go directly to the operational phase, i.e., Decision point 7. It depends on whether or not the intention is to use a research methodology or the focus is on a single research method.

Decision point 6 relates to deciding whether or not to use a research methodology, and if so which methodology. The viewpoint taken here is that a research methodology encompasses more than one research method for data collection. Here, three research methodologies are briefly described, i.e., case study research, action research, and design science.

- *Case study* is a research methodology focusing on investigating a contemporary phenomenon in its real-life context, using more than one data collection method. Case study research is a commonly used research methodology in software engineering, and guidelines for case study research in software engineering have been published, for example, by Kitchenham et al. [136], Runeson and Höst [211], Verner et al. [260], and Runeson et al. [212]. The research can be conducted using a single case study or multiple case studies, and it can be quantitative, qualitative, or employ a mixed approach, as discussed in Decision point 5. It is essential to choose an appropriate unit of analysis, i.e., based on the research questions. The unit of analysis can be a project, a software team, a specific role in an organization, or the organization as such.
- *Action research* has a lot in common with case study research. In action research, the researcher takes part in the investigation, solution development, and application of the solution. The investigation is launched based on a problem

in an organization, and the researcher studies the problem using a scientific approach. The main difference from a case study is that the researcher takes an active part in implementing the solution and that the focus is more on the change in the organization under study than on the generalized knowledge production. Thus, the researcher is much more involved in implementing the result in the organization where the research is conducted than in a case study, where the potential implementation is left to the organization. The researcher acts as a change agent in action research. Action research in a software engineering context is described in detail by Staron [242]. A potential weakness with the methodology is that the researcher becomes too involved, which creates a potential lack of objectivity of the researcher [235].

- *Design science methodology* is focused on building and evaluating artifacts that are designed to address a problem. The overall aim is to derive generalized knowledge about problems and related solutions. Design science methodology, as a problem-solving process, was introduced by Hevner et al. [106]. It requires the creation of an artifact for a specific problem. The artifact should be innovative and effective. Furthermore, it needs to be evaluated using rigorous research methods. Design science comes from information systems. Wieringa [266] helped bridge the gap from information systems to software engineering. Design science in software engineering is further explored by Runeson et al. [213] and Engström et al. [70].

  The design science methodology has similarities to action research. However, in design science, the focus is on a problem, which does not primarily come from an organization. The artifact in design science is primarily targeted at contributing to the knowledge base of researchers [30], and not at solving a practical problem.

These three methodologies are further elaborated in an industry–academia research collaboration by Wohlin and Runeson [272].

2.4 Operational Phase

The operational phase is concerned with selecting one or more appropriate research methods as a basis for data collection, and deciding which data analysis methods to use. The choices of methods should be aligned with the research question and previous decisions made, i.e., Decision points 1–6. Thus, the operational phase includes two decision points. For each of the decisions, several options are available. Figure 2.1 includes a selection of research methods and data analysis methods often applied in software engineering. One or more options may be chosen from each decision point.

According to Seaman [217], interviews and participant observation are commonly used in software engineering. Some commonly used quantitative data collection methods are archival analysis, surveys, simulation, and experiments.

There are also mixed approaches such as literature reviews, or systematic literature studies, which include systematic literature reviews and systematic mapping studies, and small-scale evaluations. There are other research methods than those included in Fig. 2.1. Further details of research methods in empirical software engineering can be obtained from the edited books by Shull et al. [226] and Felderer et al. [74].

The research methods in Fig. 2.1 can be summarized as follows:

- *Interviews* implies soliciting the viewpoints of relevant participants in relation to the research being conducted. Interviews are conducted through personal contact between the researcher and the interviewee. Interviews may be conducted through face-to-face meetings, telephone interviews, or video meetings. Advantages of interviews include the possibility to ask in-depth questions and also follow up the responses provided by the participants [217]. Interviews may be performed individually or in a focus group [149]. Interviews can be structured, semi-structured, and unstructured. Structured interviews are based on closed questions and unstructured interviews on open questions, and semi-structured interviews are a mixture.
- *Observation* or participant observation is a method to collect non-verbal data through observation of individuals and different events in a work environment. It allows the researchers to get an in-depth understanding of the environment in which the research is conducted. However, the observations need to be carefully documented in field notes [68]. When doing observations, ethical concerns are of particular importance. Ethics are discussed in Sect. 3.1.
- *Archival analysis* is a research method that collects data by extracting data from archives, for example, email records, open source projects, code repositories, or social media. It is essential to ensure that you have permission to collect the data. The method includes locating the appropriate data, collecting the data systematically, and analyzing and interpreting the data.
- *Survey* research is conducted using questionnaires and it is typically aimed at creating a broad understanding in relation to the research question since questionnaires allow many potential respondents to be contacted. Pinsonneault and Kraemer [197] list purpose of the survey, the unit of analysis, and the importance of having a representative sample as the three most important aspects of survey research. Kitchenham and Pfleeger [134] provide guidelines for conducting survey research in software engineering.
- *Simulation* is focused on building a model of some real-world entity, which then can be executed to, for example, evaluate different alternative solutions in a real-world situation. Simulations can be in three dimensions [154]. The three dimensions are: static vs. dynamic, discrete vs. continuous, and deterministic vs. stochastic simulation models. Müller and Pfahl [183] provide a discussion on the use of simulation for the software development design process or project, and bin Ali et al.[1] provide an overview of the use of process simulation in industry.
- *Systematic literature study* is a method to systematize the research literature on a topic by identifying so-called primary studies [133, 146, 147, 255]. Systematic literature studies are referred to as secondary studies. Relevant primary studies

are identified by searching the literature using either search strings in a selection of databases or snowballing based on identifying a start set of publications from which both backward and forward snowballing are conducted. Backward snowballing refers to papers being cited in an identified primary study, and forward snowballing refers to papers citing the identified primary study [268]. Two main types of systematic literature studies exist, i.e., systematic literature reviews and systematic mapping studies (sometimes called scoping studies). Systematic literature reviews are conducted to synthesize evidence in a research area, and systematic mapping studies primarily focus on structuring an area.

- *Experiment* is a controlled type of study where the objective commonly is to compare two or more alternatives. A hypothesis is formulated and the researcher would like to be able to show a cause and effect relationship based on the treatments provided to the participants. Experiments are the main focus of this book. An overview is provided in Chap. 6 and experiments are discussed in detail in Parts II and III of the book.
- *Small-scale evaluation* is a way to demonstrate or show the feasibility of a concept or tool [206]. The main objective is to illustrate the value of a proposed solution through some form of evaluation. It typically involves a single researcher or a small team of researchers. The evaluation is run over a short period of time with limited resources most often in a single site, for example, a laboratory. This type of evaluation is closely related to engineering research, where a novel solution is constructed and then assessed. The assessment may be done using other different research methods such as simulation or an experiment. We have chosen to include small-scale evaluation as a research method since far too many small-scale evaluations are labeled case studies by authors [269, 271].

The final decision point, Decision point 8, targets the selection of one or more appropriate data analysis methods. Figure 2.1 provides examples of data analysis methods often found in the software engineering literature. The data analysis methods in the figure can be summarized as follows:

- *Grounded theory* was developed by Glaser and Strauss in the 1960s. It has since evolved as described by Glaser and Strauss [87]. The method is based on analyzing data through phases of coding with the objective to identify patterns to build theories and hypotheses. Three different approaches to coding can be used, i.e., open, axial, and selective [87]. There are different variants of grounded theory and the variant most appropriate for the research question and data should be selected. According to Stol et al. [246], several articles claiming to use grounded theory in software engineering do not include all core characteristics of grounded theory. Stol et al. [246] and Hoda [107] also provide guidelines for conducting grounded theory in software engineering.
- *Thematic analysis* is a qualitative data analysis technique, the objective of which is to provide a deep insight concerning the data content. According to Braun and Clarke [35], thematic analysis is a method to analyze, identify, and report themes within the data. The method involves open coding, i.e., the codes emerge as the data is analyzed. They provide guidelines for performing thematic analysis.

Thematic analysis preferably includes several researchers to allow the themes to be discussed as they emerge. Thematic analysis has commonalities with grounded theory, although the objective is different.

- *Content analysis* is used to analyze documents, including emails, or communication artifacts, for example, videos, pictures, and recorded material. It is intended to be replicable since others may analyze the same documents or artifacts. An advantage with content analysis is that it is non-intrusive. In a similar way as for grounded theory and thematic analysis, the findings are coded or labeled. An overview of content analysis is provided by Harwood and Garry [102], and a more in-depth description of content analysis can be found in the book by Drisko and Maschi [61].
- *Classification analysis* is a method to assign a class to, for example, a study of a concept, typically with predefined classes. This is illustrated by the work by Jiménez and Piattini [120] who used it to classify processes in the software life cycle. Classification analysis is also used as a tool to analyze large amounts of data automatically in which case classes may be identified through learning. Thus, it is also a data mining technique, for example, for text analysis. It is intended to help researchers to get an increased understanding of their data sets, and hence make more informed predictions. Krishnaiah et al. [151] present a survey of classification techniques related to data mining.
- *Statistical analysis* is used to analyze quantitative data. The data can either be collected as quantitative data or converted into quantitative data, for example, through frequency counts of items. The items can, for example, be people, events, themes, or words. Two main forms of statistical analysis exist, i.e., descriptive and inferential analysis. Descriptive analysis is a way of summarizing the data. It includes presenting, for example, mean values and variation as well as different types of plots, including box plots. Inferential analysis includes, for example, testing of hypotheses, regression analysis, and other forms of statistical analysis to identify relationships. Furthermore, statistical methods may be non-parametric or parametric. Non-parametric methods do not make assumptions on the distribution of the data, while parametric methods make an assumption concerning the distribution of the data. If the assumption regarding the distribution is correct then parametric analysis is more powerful than non-parametric. Statistical analysis of data from experiments is further discussed in Chap. 11. Numerous books on statistics are available in the literature.

Given decisions on the decision points in Fig. 2.1, a research design has been created, and the study can be run. Care should be taken when performing the study to minimize risks and threats to the research. Validity for experiments is discussed in Sects. 9.7–9.9, although the discussion on different types of validity threats is also applicable to other types of research. When the study has been conducted, the research findings can be presented as shown in the lower right corner of Fig. 2.1.

2.5 Example Decision-Making Structure

To illustrate the decision-making structure, an example is presented. Assume that we would like to compare pair programming with solo programming, and evaluate the potential benefits with pair programming. We decide that we would like to evaluate three aspects, i.e., faster, better, and cheaper. Faster is about the time it takes in calendar time, better relates to the quality produced, and cheaper is a matter of the total time spent, for example, in hours. When it comes to the evaluation, we may have a hypothesis that a pair may be faster and produce higher quality but probably not cheaper. A hypothesis is a conjecture that can be evaluated empirically. Thus, we would like to evaluate if this is correct or there is some other outcome. Our overall research question (RQ) is formulated as follows:

RQ – How good is pair programming in relation to solo programming when comparing them concerning faster, better, and cheaper?

The overall research question is turned into an overarching hypothesis, i.e., pair programming is equally good as solo programming. It should be noted that it requires that all three aspects in the overall research question are evaluated. As a result of the research, we would like to reject the hypothesis that they perform equally, i.e., to understand which way of programming performs best. Hypothesis formulation is further discussed in Sect. 9.2.

This leads to Decision point 1. Given that we would like to understand the benefits of pair programming versus solo programming, the research is *basic* since we do not have any intention to improve some existing practice. Next, we turn to the research logic (Decision point 2). We want to collect measurements to compare the two ways of performing programming. Thus, the research is *deductive* since our starting point is a hypothesis that we would like to evaluate. In relation to Decision point 3, given our focus on evaluating and comparing two ways of programming, we conclude that our focus is on *evaluation research*. Decision point 4, the final decision point in the strategy phase, is concerned with the research paradigm applied. Our research can be classified as *positivist* research since we assume in our hypothesis that solo programming is inherently faster, better, or cheaper. It is here also noted that we should try to document our study in such a way that it can be replicated.

After finalizing the four first decision points, we turn to the tactical phase. Given the focus on evaluation with the objective to compare pair programming and solo programming, we conclude, in Decision point 5, that our research process is *quantitative*. We need to collect different measures to be able to evaluate both time and quality. For the latter, we decide to focus on quality in terms of the number of defects. However, different measures do not imply that we need different data collection methods as such. Thus, we conclude that we do not need a research methodology to answer the research question, which means that we continue to the operation phase without further investigation of different research methodologies, i.e., Decision point 6.

Next, we turn to Decision point 7, where we have to decide upon an appropriate research method to answer our research question. Given that we want to compare and evaluate quantitatively, we choose to conduct an *experiment*. At this stage, we should design our experiment. Finally, we would like to perform *statistical analysis*, Decision point 8, to compare the two programming alternatives. For how to design and analyze an experiment, we refer to Chap. 6 and Parts II–III.

Pair programming experiments can be found in, for example, Williams and Kessler [267] and Arisholm et al. [8], and a systematic literature review on the topic is presented by Hannay et al. [101]. Some further examples related to decision-making involving other research methodologies and research methods can be found in Wohlin and Aurum [270].

2.6 Research Approach Comparison

The prerequisites for an investigation limit the choice of research approach. Four different research approaches are compared to highlight the differences among the chosen approaches. These four approaches are discussed in Chaps. 4–7. Experiments are then discussed in detail in Parts II–III. A similar comparison can be done for the other research approaches, for example, those in Fig. 2.1. The comparison of approaches can be based on a number of different factors. Table 2.1 is an extension of the different factors discussed by Pfleeger [194]. The factors are further described below.

Execution control describes how much control the researcher has over the study. For example, in a case study, data is collected during the execution of a project. If management decides to stop the studied project due to, for example, economic reasons, the researcher cannot continue to carry out the case study. The opposite is the experiment where the researcher is in control of the execution.

Measurement control is the degree to which the researcher can decide upon which measures to be collected, and which measures to include or exclude during execution of the study. An example is how to collect data about requirement volatility. During the execution of a survey we cannot include these kinds of measures, but in a case study or in an experiment it is possible to include them.

Table 2.1 Empirical research factors

Factor	Systematic literature study	Survey	Experiment	Case study
Execution control	Some	No	Yes	No
Measurement control	Yes	No	Yes	Yes
Investigation cost	Medium	Low	High	High
Ease of replication	High	High	High	Medium

In a survey, we can only collect data regarding people's opinions about requirement volatility.

Closely related to the factors above is the *investigation cost*. The cost differs depending on which approach is chosen. This is related to, for example, the size of the investigation and the need for resources. The approach with the lowest cost is the survey, since it does not require a large amount of personnel resources. Systematic literature studies can be labor intensive. However, the researcher is in control and could potentially adapt the study to not make it too costly. The difference between case studies and experiments is that if we choose to investigate a project in a case study, the outcome from the project is some form of product that may be retailed, i.e., it is a real-life investigation. In an offline experiment the outcome is some form of experience or knowledge which is not directly profitable in the same way as a product.

Another important aspect to consider is the possibility to *replicate* the investigation. The purpose of a replication is to show that the result from, for example, the original experiment is valid for a larger population. A replication becomes a "true" replication if it is possible to replicate both the design and the results. It is not uncommon that the objective is to perform a replication, but the results, to some extent, turn out differently than the results of the original study. Replication is further discussed in Sect. 3.2.

Another aspect related to replication, in the sense that it is concerned with studies over time, is longitudinal studies [203]. The main difference between a longitudinal study and a replication is that a longitudinal study is primarily conducted with the same subjects and a replication is mostly a study conducted with new subjects. In other words, replication means several studies and a longitudinal study is a single study. The longitudinal study is conducted over a period of time; for example, a survey can be done on several occasions, experiments can be repeated, and the case study may also be longitudinal if it is conducted over a period of time. A longitudinal study is normally conducted to understand, describe, or evaluate something that changes over time [207].

The choice of empirical approach depends on the prerequisites for the investigation, the purpose of it, available resources, and how we would like to analyze the collected data. Easterbrook et al. [65] provide more advice on selection of research approaches.

Furthermore, the borderline between different types of study is not always clear cut. For example, a comparative case study may also be referred to as a quasi-experiment in an industrial context, and a post hoc observational study of software engineering course outcomes may also be referred to as a student experiment.

Systematic literature studies are common in software engineering as they are aimed at providing a summary and overview of a defined research area in software engineering. The execution control is limited since the research method uses existing publications, while the measurement control is high.

Table 2.2 Design type and qualitative vs. quantitative data in empirical strategies

Methodology/Method	Design type	Qualitative/Quantitative
Systematic literature study	Fixed	Both
Survey	Fixed	Both
Experiment	Fixed	Foremost quantitative
Case study	Flexible	Both

Surveys are very common within social sciences where, for example, attitudes are polled to determine how a population will vote in the next election. A survey provides no control of the execution or the measurement, though it is possible to compare it with similar ones, but it is not possible to manipulate variables as in the other investigation methods [13].

An experiment is a formal, rigorous, and controlled investigation. In an experiment the key factors are identified and manipulated, while other factors in the context are kept unchanged (see Sect. 6.2). The separation between case studies and experiment can be represented by the level of control of the context [190]. In an experiment, different situations are deliberately enforced and the objective is normally to distinguish between two situations, for example, a control situation and the situation under investigation. Examples of the manipulated factors could be, for example, the inspection method or experience of the software developers. In a case study, the context is governed by the actual project under study.

Some of the research approaches could be informed by qualitative or quantitative data, depending on the design of the investigation (see Table 2.2). In a systematic literature study, some data collected from the identified studies to include may be quantitative and others qualitative. It depends heavily on what has been reported in the included studies.

The classification of a survey depends on the design of the questionnaires, i.e., which data is collected and if it is possible to apply any statistical methods. This is also true for case studies but the difference is that a survey is done in retrospect while a case study is done while a project is being executed. A survey could also be launched before the execution of a project. In the latter case, the survey is based on previous experiences and hence conducted in retrospect to these experiences, although the objective is to get some ideas about the outcome of the forthcoming project.

Experiments are foremost quantitative since they have a focus on measuring different variables, changing them, and measuring them again. During these investigations quantitative data is collected and then statistical methods are applied. However, qualitative data may be collected to help with interpretation of the data [129].

2.7 Empirical Evaluation of Process Changes

To illustrate how empirical research approaches (research methodology or research method) can be used in an overarching research initiative, for example, a research project or during PhD studies, the four research approaches discussed in the previous section are here presented in the context of changing the way software development is performed. It can be the change of part of the software development process or some specific action or activity. To simplify the discussion, all different types of changes are here called process change.

An improvement-seeking organization wants to assess the impact of process changes (e.g., a new method or tool) before introducing them to improve the way of working. Empirical studies are important to get objective and quantifiable information on the impact of changes. In the previous section, four empirical approaches are compared. This section describes how the approaches may be used when software process changes are evaluated [273]. The objective is to discuss the approaches in terms of a suitable way of handling technology transfer from research to industrial use. The selection of a research methodology for research in collaboration between industry and academia is discussed by Wohlin and Runeson [272]. Technology transfer and some different steps in that process in relation to using empirical research are discussed in further detail in Sect. 3.6.

In Fig. 2.2, the approaches are placed in appropriate research environments. The order of the approaches is based on the "normal" size of the study. The objective is to order the studies based on how they may typically be conducted to enable a controlled way of transferring research results into practice. Systematic literature studies focus on the literature and they provide a background to the knowledge in the area of the research. They may help in identifying a suitable improvement. Furthermore, knowledge of an area, for example obtained through the systematic literature study, may help in formulating questions in a survey. The survey may either be targeting software developers generally or the developers in the organization wanting to improve their software development. A survey does not intervene with the software development to any large extent, which means that it is

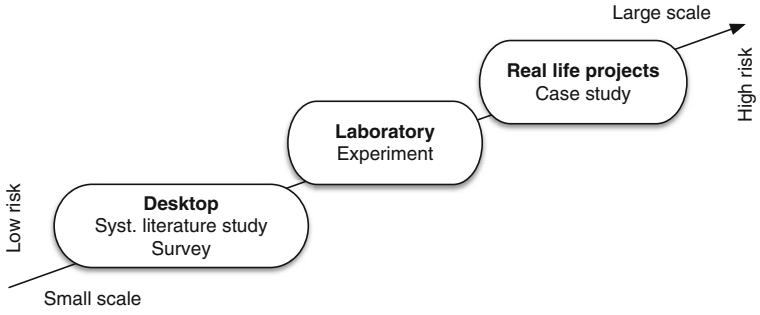


Fig. 2.2 Systematic literature studies, surveys, experiments, and case studies

a low-risk research method. Next, it is possible to run an experiment. It is mostly rather limited in comparison to a real project and the case study is typically aimed at one specific project or activity. Moreover, an experiment may be carried out in a university environment prior to doing a study in industry, hence lowering the cost and risk of the process change (see also Linkman and Rombach [160]). Thus, the different research approaches help build knowledge of an area before introducing something novel into a real-life development organization. An overview of the four research approaches in Fig. 2.2 can be found in Chaps. 4–7 in the order that they appear in the figure.

The research environments for the four research approaches are as follows:

Desktop — Knowledge of an area may be built by studying the research literature through a systematic literature study. As a potential change proposal is identified, it is possible to solicit opinions about the proposal using a survey. Thus, the change proposal is evaluated offline without executing the changed process. Hence, this type of evaluation does not involve people who apply the method, tool, etc.

Laboratory — The change proposal is evaluated in an offline laboratory setting (in vitro[3]), where an experiment is conducted and a limited part of the software development process is executed in a controlled manner.

Real life — The change proposal is evaluated in a real-life development situation, i.e., it is observed online (in vivo[4]). This involves, for example, pilot projects. In this environment it is often too expensive to conduct controlled experiments. Instead, case studies are often more appropriate.

In Fig. 2.2, the placement of the different research environments indicates an increase in scale and risk. In order to try out, for example, a new design method in a large-scale design project and in a realistic environment, we may apply it in a development project as a pilot study. This is, of course, more risky compared to a laboratory or desktop study, as failure of the process change may endanger the quality of the delivered product. Furthermore, it is often more expensive to carry out experiments and case studies, compared to desktop evaluation, as a desktop study does not involve the execution of a development process. It should be noted that the costs refer to the cost for investigating the same thing. For example, it is probably less costly to first interview people about the expected impact of a new inspection (or review) method than performing a controlled experiment, which in turn is less costly than actually using the new method in a project with the risks involved in adopting new technology.

Before a case study is carried out in a development project, limited studies in either a desktop or laboratory environment or both should be carried out to reduce risks. However, there is no general conclusion on order and cost; for every change proposal, a careful assessment should be made of which empirical approaches are

[3] Latin for "in the glass" and refers to chemical experiments in a test tube.

[4] Latin for "in life" and refers to experiments in a real environment.

most effective for the specific situation. The key issue is to choose the best approach based on cost and risk, and in many cases it is recommended to start at a small scale and then scale up as the knowledge increases, which also means that the risk decreases as the study is scaled up.

Independently of which research approach we use, there is a need for methodology support in terms of how to work with improvement, how to collect data, and how to store the information. These issues are further discussed in several of the sections in Chap. 3 and an overview of the four research approaches is provided in Chaps. 4–7.

2.8 Concluding Remarks

It is essential to select an appropriate research approach in relation to the research question. For example, an experiment should be chosen when the objective is to compare two or more alternative ways of doing some activity. However, it is not only essential to make a good selection; it is also crucial to ensure that research studies are labeled correctly. If claiming to conduct, for example, an experiment, the study should adhere to *the definition* of being an experiment. Thus, when selecting a research approach, researchers should also ensure that they adhere to definitions and expectations for the chosen research approach. It is recommended to follow published guidelines for the selected research approach after having ensured that the study is labeled correctly, i.e., adhere to the definition of the research approach.

Unfortunately, several studies are mislabeled or researchers do not follow the expectations and guidelines for the type of research approach they claim to use. Several authors have found that research studies in software engineering are published and claimed to use a specific research approach, but several studies are either mislabeled or do not follow published guidelines. The following three types of studies are examples of problems identified in the literature:

- *Experiment*: Ayala et al. [12] investigated the use of the label experiment in the area of mining software repositories. They conclude that the label "experiment" is often not used according to the definition. In their sample of studies, close to 20% of the studies were not experiments; they should rather be labeled as observational studies. Moreover, for the remaining studies, only one study is a genuine controlled experiment. The other studies can be labeled experiment, but these experiments are with limited control. For example, they can be quasi-experiments as discussed in Sect. 6. The latter contradicts one of the expectations of a genuine experiment.
- *Case study*: Already in 2006, Zannier et al. [283] observed that "... our sample indicated a large misuse of the term case study." Similarly, Runeson and Höst conclude "...the presented studies range from very ambitious and well organized studies in the field, to small toy examples that claim to be case studies" [211]. To investigate whether these conclusions have changed over the years, Wohlin [269]

and Wohlin and Rainer [271] conducted studies concerning the use of the label "case study." Both studies concluded that about 50% of the claimed case studies are not case studies according to the definitions. The most common misclassification of case studies is that small-scale evaluations are classified as being case studies. However, a small-scale evaluation does not fulfill the definitions for being a case study. It is not conducted in a real-life context and is not studying a contemporary phenomenon.

- *Grounded theory*: Stol et al. [246] observed that many studies in software engineering claiming to use grounded theory do not include all core characteristics of grounded theory. Thus, the studies are not using grounded theory according to the expectation of such a study.

Mislabelling or not following guidelines becomes a problem when, for example, conducting a systematic literature study and looking for a specific type of study. Furthermore, it is confusing for researchers and practitioners when trying to interpret research findings. A practitioner may be interested in studies conducted in a real-life context (from their perspective), but when looking for case studies, many studies that are not case studies are found. Thus, the confidence in the research is affected.

2.9 Exercises

2.1 Describe the differences between exploratory, descriptive, explanatory, and evaluation research.

2.2 What is the difference between qualitative and quantitative research?

2.3 How would you judge the four factors in Table 2.1 for the research methods *observation* and *small-scale evaluation* in Fig. 2.1?

2.4 How can different research approaches be used in a research program, for example, during PhD studies?

2.5 Why is a correct labelling of research studies essential?

Chapter 3
Essential Areas in Empirical Research

Several areas are essential in empirical software engineering research independently of the research approach chosen. This chapter introduces four areas to consider carefully when conducting empirical research, presented in Sects. 3.1–3.4. First, ethics is discussed in Sect. 3.1. Empirical research often involves human subjects, and hence ethical considerations are crucial. Replicability of studies is essential to the knowledge-building process. Replication of studies is elaborated in Sect. 3.2. As knowledge is acquired through empirical research, it may lead to being able to formulate one or more theories. Theories in software engineering are the topic of Sect. 3.3. The fourth area concerns measurements, which is essential as part of the data collection in an empirical study. Measurement is discussed in Sect. 3.4. The chapter concludes with two sections concerned with improving software development. Section 3.5 introduces several concepts that are important when aiming to improve software development. Finally, a model for technology transfer to industry based on empirical research is summarized in Sect. 3.6.

3.1 Ethics

Any empirical research activity involving human subjects must take ethical aspects into consideration. Some aspects are regulated by national laws; others are not regulated at all. Andrews and Pradhan identified ethical issues in software engineering, and found existing policies to be insufficient [4]. Hall and Flynn surveyed ethical practice and awareness in the UK, and found alarming unawareness [98], and nothing indicates that the UK is an exception.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_3

Singer and Vinson initiated a discussion on ethical issues [230], continued to discuss cases of ethical issues [231], and provided practical guidelines for the conduct of empirical studies [262]. They identified four key principles:

- Subjects must give *informed consent* for their participation, implying that they should have access to all relevant information about the study, before making their decision to participate or not. Their decision must be explicit and free, also with respect to implicit dependencies on managers, professors, etc.
- The study should have *scientific value* to motivate subjects to expose themselves to the risks of the empirical study, even if these are minimal.
- Researchers must take all possible measures to maintain *confidentiality* of data and sensitive information, even when this is in conflict with the publication interests.
- Weighing risks, harms, and benefits, the *benefits* must overweigh, not only for the individual subjects, but also for groups of subjects and organizations.

These principles are turned into more practical guidelines below, related to planning, conduct, and reporting of an experimental study. We also refer to Sieber [227] for a checklist of risks for subjects to be addressed in experimentation and to Strandberg [248] for ethical aspects of interviews.

Ethical Review In countries where legislation requires an ethical review for studies involving human subjects, like Canada, the USA, and Australia, the procedures and documentation for such studies have to be followed to enable the study. The review implies a proposal being put before an Ethical Review Board (ERB) at the university or government agency, for approval. These procedures are mostly derived from the needs in biomedical research, and are thus generally not tailored to software engineering needs. Vinson and Singer mention, for example, that in Canada, it is not clear whether studies using source code (being written by humans and revealing information about them) and its data are subject to the review procedures [262].

The documentation needed in the review typically includes a description of the project, comprising details on subjects and treatments, documentation of how informed consent is obtained, and a review of ethical aspects of the project.

Informed Consent The basis for a human-oriented empirical study (e.g., an experiment) is that subjects are participating voluntarily, and that they have enough information to make the decision to participate or not. Further, this includes the option to withdraw from the study at any time, without any penalty for the subject. In order to make this decision process clear and explicit, consent should be given in writing.

A consent form typically comprises the following elements [262]:

- *Research project title:* for identification purposes.
- *Contact information:* both research and ethics contact.
- *Consent and comprehension:* the subjects state that they understand the conditions for the project and accept them.
- *Withdrawal:* states the right to withdraw without penalties.

- *Confidentiality:* defines the promises about confidential handling of data and participation.
- *Risks and benefits:* explicitly listing what the subjects risk and gain.
- *Clarification:* the right for the subject to ask questions for clarification of their role in the study.
- *Signature:* mostly by both subject and researcher, one copy for each, to indicate it is a mutual agreement.

In some experimental designs, *full disclosure* of the research goal and procedures may compromise the conduct of the experiment as such. For example, knowing the hypothesis beforehand, the subjects may change their behavior accordingly. Then, *partial disclosure* may be used, meaning that the experimental goals and procedures are presented at a higher level of abstraction.

For empirical studies in companies (in vivo), the consent must include both the organization and the individual subjects. In particular, the subjects cannot be ordered to participate, and are free to withdraw without penalties. Further, issues of confidentiality and sensitive results *within* the company also must be taken into consideration.

The consent may be differentiated on whether it is given for the goals of the current study, or if data may be used for further studies with different goals.

Confidentiality The subjects must be sure that any information they share with researchers will remain confidential. The three aspects of confidentiality are [262] as follows:

- *Data privacy*, referring to restricted access to data, imposed by, for example, password protection and encryption
- *Data anonymity*, addressed by keeping the identities of subjects apart from the data
- *Anonymity of participation*, meaning that the consent decision should be kept secret

Since the empirical studies (including experiments) aim at drawing general conclusions, there is no principal conflict with keeping the specifics confidential. Data privacy issues can also be solved by good working practices. However, as the number of subjects is often small, there is a risk that information may be traced to individuals, even if anonymized, thereby threatening anonymity. Moreover, for the external validity of the study (see Sect. 9.7), information about the study context should be reported, which may conflict with the anonymity.

The anonymity of participation is the hardest to achieve. Students in a class who are enrolled in experiments may have the formal right to decline participation, but it is hard to hide from the researcher which students participate or not. Similarly, in companies, managers would easily know who is participating in the study. Vinson and Singer advice that “for studies involving students, researchers should avoid recruiting students in the classroom setting and should avoid trying to recruit their own students” [262], advice followed by few.

Sensitive Results Outcomes from any empirical study may be sensitive in different respects for different stakeholders. The individual performance of a subject is one example which managers or professors would like to see. The conclusions from the empirical study may also be sensitive, particularly if a sponsor of the project has a stake in it. The results may also be sensitive to the researchers, for example, if an experiment does not support their hypotheses.

These situations stress the moral standards of the stakeholders and require balancing public interests versus those of the stakeholders [167]. Possible measures to take to prepare for these situations include different kinds of independence. For results sensitive to:

- *Subjects*, make sure that confidentiality procedures apply, independently of facts revealed (crime exempted [231]).
- *Sponsors*, include clear statements on rights for independent publications of the anonymized results in the informed consent form for companies, and in research project contracts [167].
- *Researchers*, consider having peers to perform statistical analyses on anonymized data (both subjects and scales) independently from the experimenters, particularly when the treatment is designed by the experimenters themselves. This also reduces the threat of experimenter expectancy.

These actions reduce the risk of being stuck in ethical dilemmas, and increase the validity of all empirical studies.

Inducement In recruiting subjects for an experiment, there must be inducements to motivate their participation. The experience and knowledge gained by applying a new method may be inducement enough. In order to treat all participants fairly, all subjects should be given the opportunity to learn about all treatments, even if the experimental design does not require it.

Some monetary inducement may also be involved, for example, in the form of cash payment, participation in a lottery, or, for professional subjects, their ordinary salary. Independently of form, the inducement must be balanced to ensure that the consent to participate really is voluntary, and not forced by too large economic or other inducements.

Feedback To maintain long-term relationships and trust with the subjects of a study, feedback on results and analysis is important. Subjects must not agree on the analysis, but should be given the opportunity to get information about the study and its results. If feasible, from a confidentiality point of view, data from an individual's performance may be reported back together with the overall analysis.

Conclusion on Ethics Singer and Vinson ask in their early work for a code of ethics for empirical software engineering [231]. Still, 10 years later, the community has not yet developed one; the closest is Vinson and Singer's guidelines [262], which are summarized above. Research funding agencies start to require that general codes of ethics be applied, which may not fit the purpose. Concrete and tailored ethical guidelines for empirical software engineering research would benefit both

the subjects which they aim to protect and the development of the research field as such.

3.2 Replications

The replication of an experiment involves repeating the investigation under similar conditions, while for example, varying the subject population. This helps in finding out how much confidence it is possible to place in the results of the experiment. If the assumption of randomization is correct, i.e., the subjects are representative of a larger population, replications within this population show the same results as the previously performed experiment. If we do not get the same results, we have been unable to capture all aspects in the experiment design that affect the result. Even if it is possible to measure a certain variable or to replicate an experiment, it might be difficult and too costly.

Replications may be of different types [123, 226]:

- *Close* replications follow the original procedures as closely as possible. This type is sometimes referred to as *exact* replications [226].
- *Differentiated* replications study the same research questions, using different experimental procedures. They may also deliberately vary one or more major conditions in the experiment.

Basili et al. [27] proposed a more fine-grained classification:

- Strict replications (synonym for close and exact)
- Replications that vary variables intrinsic to the study
- Replications that vary variables intrinsic to the focus of the study
- Replications that vary the context variables in the environment in which the solution is evaluated
- Replications that vary the manner in which the experiment is run
- Replications that extend the theory

In their work on artifact review and badging in software engineering research, the Association for Computing Machinery (ACM) uses the following definitions, which are primarily defined in a technology-oriented experimental context [9]:

- *Repeatability*: Same team, same experimental setup
- *Reproducibility*: Different team, same experimental setup
- *Replicability*: Different team, different experimental setup

ACM has defined a set of badges that are assigned to research studies in ACM conferences and journals to signal whether research artifacts are available, and if research results are reproduced or replicated [9].

In other fields of research, many different classification schemes are used [90], and no standardized terminology exists across fields of research. Neither is the terminology in the software engineering field established. The above-presented distinction between *close* and *differentiated* replications is a starting point for specifying replications in software engineering.

The advantage of close replications is that the known factors are kept under control, building confidence in the outcome. However, close replications sometimes require the same researchers to conduct the study, as they have tacit knowledge about the experiment procedures which are difficult to document [224, 225]. On the other hand, there is a substantial risk for experimenter bias in close replication studies [132]. Furthermore, it is questioned whether any replication in software engineering may be classified as close, since so many factors may vary in the complex setting of a software engineering experiment [123].

Differentiated replications on the other hand may be used for more exploratory studies. Arguments are raised for replicating original hypotheses, rather than specific experimental designs [175], i.e., in favor of differentiated replications rather than close replications.

To enable replications, closed or differentiated, factors and settings must be well documented and analyzed, thus providing more knowledge from replicated studies. Factors to consider and report for replication studies include [123] the following, which are discussed in detail in Chap. 9:

- *Site* where the experiment is conducted
- *Experimenters* conducting the experiment
- *Design* chosen for the experiment
- *Instrumentation*, i.e., forms and other material
- *Variables* measured
- *Subjects* conducting the experiment

In empirical software engineering, experimental material has been made available as replication or laboratory packages [237] to some extent. These are used as a foundation for families of experiments [216], where experiments are systematically replicated to gain more knowledge or validate previous knowledge. Shepperd et al. conclude in their analysis of 28 replications of 35 original experiments that there is a need for better reporting of both original and replicated studies [221]. Given the lack of information and low power of experiments, it can be unclear whether a replication is confirmatory and to what extent the experiment adds to the knowledge of the software engineering community. Thus, we do not even know if there is a "replication crisis" in software engineering.

A more general movement toward improving research through transparency is *Open Science* as defined by UNESCO [254] to include open scientific publications, open research data, open educational resources, open source software, and open hardware. Thereby, both replications and re-analysis of existing experiments are supported. Mendez et al. explore implications of open science for software engineering research [169], Runeson et al. provide guidelines for open data and artifacts [214], and the *Empirical Software Engineering* journal has defined an open science policy [171]. Particularly with respect to sharing experimental material and data, they stress the need for permanent digital archival resources, linked to universally accessible digital object identifiers (DOI).

Open science, aiming to increase research validity through transparency, may be in conflict with confidentiality for the study participants. Not all research data can

be published openly, even with participant consent [214]. Therefore, we promote FAIR data rather than open data, meaning that data should be "as open as possible and as closed as necessary". FAIR—Findable, Accessible, Interoperable, Reusable Data[1]—focuses on making research data possible to find through meta-data, while the accessibility also takes confidentiality into account. Data may be shared under a specific license for special purposes—e.g., meta-analysis—rather than made fully accessible.

3.3 Theory in Software Engineering

"A theory provides explanations and understanding in terms of basic concepts and underlying mechanisms, which constitute an important counterpart to knowledge of passing trends and their manifestation" [100]. Experiments may be conducted to generate, confirm, and extend theories, as mentioned above. However, the use of theory is scarce in software engineering, as concluded by Hannay et al. in their systematic literature review of software engineering experiments, 1993–2002 [100]. They found 40 theories in 23 articles, out of the 113 articles in the review. Only two of the theories were used in more than one article! Similarly, Hall et al. identified a lack of references to general theory on motivation in 92 studies of motivation in software engineering, in the 1980–2006 period [99].

Endres and Rombach [69] identify a list of 50 findings which they refer to as "laws," which is a notion for a description of a repeatable phenomenon in a natural sciences context. Endres and Rombach apply this notion to software engineering. Many of the listed "laws" are more general than software engineering, for example, "it takes 5000 hours to turn a novice into an expert." In their notion, *theories* explain the "laws," *hypotheses* propose a tentative explanation for why the phenomenon behaves as observed, while a *conjecture* is a guess about the phenomenon. Endres and Rombach list 25 hypotheses and 12 conjectures appearing in the software engineering literature.

Zendler [285] takes another approach, and defines a "preliminary software engineering theory," composed of three fundamental hypotheses, six central hypotheses, and four elementary hypotheses. There is a hierarchical relation between the hypotheses, the fundamental being the most abstract, and elementary the most concrete ones, originating from outcomes of experimental studies. Wohlin et al. [276] present a general theory of software engineering concerning the balancing of social, human, and organizational capitals.

Gregor [96] describes five general types of theory, which may be adapted to the software engineering context [100]:

1. *Analysis:* Theories of this type describe the object of study, and include, for example, taxonomies, classifications, and ontologies.

[1] https://www.go-fair.org/fair-principles/

2. *Explanation:* These theories explain something, for example, why something happens.
3. *Prediction:* These theories aim at predicting what will happen, for example, in terms of mathematical or probabilistic models.
4. *Explanation and prediction:* These theories combine types 2 and 3, and are typically what is denoted as "empirically based theory."
5. *Design and action:* These are theories that describe how to do things, typically prescriptive in the form of design science [106]. It is debated whether this category should be denoted theory at all.

Sjøberg et al. [236] propose a framework for software engineering theories, comprising of four main parts:

- Constructs
- Propositions
- Explanations
- Scope

The *constructs* are the entities in which the theory is expressed, and to which the theory offers a description, explanation, or prediction, depending on the type of theory as defined above. *Propositions* are made up of proposed relationships between the constructs. The *explanations* originate from logical reasoning or empirical observations of the propositions, that is, the relationship between the constructs. The *scope* of the theory defines the circumstances under which the theory is assumed to be applicable. Sjøberg et al. [236] suggest the scope is expressed in terms of four archetype classes: actor, technology, activity, and software system (see Table 3.1).

Stol and Fitzgerald presented a research cycle for theory-oriented software engineering, illustrating the interplay between empirical observations and theory building [245]. Figure 3.1 illustrates that empirical observations and generalization belong to the empirical level, i.e., are conducted in the context of study, using empirical methods. The formal theory formation, which is a conceptualization of the empirical findings, belongs to the theoretical level, as well as the interpretation, explanation, and prediction based on the empirical results. This part is an analytical endeavor. A research cycle iterates over the two levels, and guidelines for practice may be derived based on the formal theory.

Table 3.1 Framework for software engineering theories, as proposed by Sjøberg et al. [236]

Archetype class	Subclasses
Actor	Individual, team, project, organization, or industry
Technology	Process model, method, technique, tool, or language
Activity	Plan, create, modify, or analyze (a software system)
Software system	Software systems may be classified along many dimensions, such as size, complexity, application domain, business/scientific/student project or administrative/embedded/real time, etc.

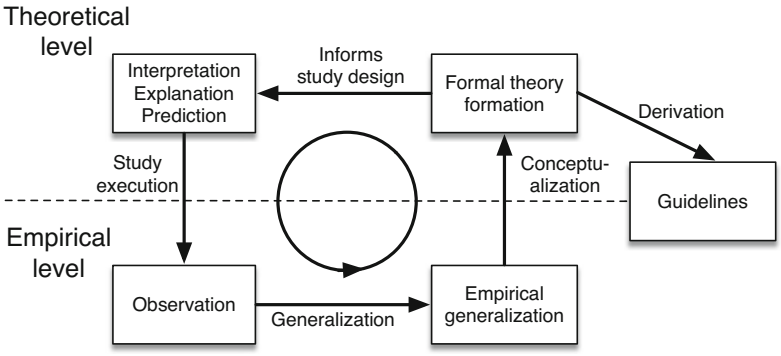


Fig. 3.1 Overview of a theory-oriented research process. (The figure is adapted from Stol and Fitzgerald [245])

Despite being attractive from a theoretical point of view, neither of these proposed theory systems has had any major impact on the software engineering field so far. Theories are important for the conceptualization and communication of knowledge within a field of research, and are useful when aggregating existing research and setting up replication studies. Theories may also be used for communication with practitioners in decision-making, whether it be strategic choices of technology or project decisions based on prediction models. Hence, theory building in software engineering should be developed for the field to develop into a mature field of science.

3.4 Measurement

Software measurement is crucial to enable control of projects, products, and processes, or as stated by DeMarco: *"You cannot control what you cannot measure"* [56]. Moreover, measurement is a central part in empirical studies. Empirical studies are used to investigate the effects of some input to the object under study. To control the study and to see the effects, we must be able to both measure the inputs to describe what causes the effect on the output, and to measure the output. Without measurements, it is not possible to have the desired control and therefore an empirical study cannot be conducted.

Measurement and measure are defined as follows [76]: "Measurement *is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.*" A measure is the number or symbol assigned to an entity by this relationship to characterize an attribute.

Instead of making a judgment directly on the real entity, we study the measures and make the judgment on them. The word metric or metrics is also often used

in software engineering. Two different meanings can be identified. First of all, software metrics is used as a term for denoting the field of measurement in software engineering. The book by Fenton and Pfleeger [76] is an example of this. Secondly, the word metric is used to denote an entity which is measured; for example, lines of code (LOC) is a product metric. More precisely, it is a measure of the size of the program. Software measurement is also further discussed by Shepperd [220].

In this section, measurement theory is presented. Sections 3.4.1–3.4.4 describe the basics of measurement theory. Examples of measures in software engineering and the relation to the statistical analysis are presented in Sect. 3.4.5, while practical aspects of measurements are discussed in Sect. 3.4.6.

3.4.1 Basic Concepts

A measure is a mapping from the attribute of an entity to a measurement value, usually a numerical value. Entities are objects we can observe in the real world. The purpose of mapping the attributes into a measurement value is to characterize and manipulate the attributes in a formal way. One of the basic characteristics of a measure is therefore that it must preserve the empirical observations of the attribute [77]. That is, if object A is longer than object B, the measure of A must be greater than the measure of B.

When we use a measure in empirical studies, we must be certain that the measure is valid. To be *valid*, the measure must not violate any necessary properties of the attribute it measures and it must be a proper mathematical characterization of the attribute.

A valid measure allows different objects to be distinguished from one another, but within the limits of measurement error, objects can have the same measurement value. The measure must also preserve our intuitive notions about the attribute and the way in which it distinguishes different objects [136]. A measure must be valid both analytically and empirically. Analytical validity of a measure relates to its ability to capture accurately and reliably the item of interest. Empirical validity (sometimes referred to as statistical or predictive ability) describes how well, for example, a score correlates to something measured in another context.

Effect size is a simple way of quantifying the difference between two groups. This is particularly important in experimentation, since it may be possible to show a statistically significant difference between two groups, but it may not be meaningful from a practical point of view. In most cases, it is possible to show statistically significant differences with a sufficiently large number of subjects in an experiment, but it does not necessarily mean that it is meaningful from a practical point of view. It may be the case that the difference is too small or the cost to exploit the difference is simply too high.

The mapping from an attribute to a measurement value can be made in many different ways, and each different mapping of an attribute is a *scale*. If the attribute is the length of an object, we can measure it in meters, centimeters, or inches, each of which is a different scale of the measure of the length.

As a measure of an attribute can be measured in different scales, we sometimes want to transform the measure into another scale. If this transformation from one measure to another preserves the relationship among the objects, it is called an admissible transformation [76]. An *admissible transformation* is also called *rescaling*.

With the measures of the attribute, we make statements about the object or the relation between different objects. If the statements are true even if the measures are rescaled, they are called *meaningful*, otherwise they are *meaningless* [38]. For example, if we measure the lengths of objects A and B to 1 meter and 2 meters respectively, we can make the statement that B is twice as long as A. This statement is true even if we rescale the measures to centimeters or inches, and is therefore meaningful. As another example, we measure the temperature in room A and room B to 10 °C and 20 °C, respectively, and make the statement that room B is twice as warm as room A. If we rescale the temperatures to the Fahrenheit scale, we get the temperatures 50 °F and 68 °F. The statement is no longer true and is therefore meaningless.

Different scale types can be defined depending on which admissible transformation can be made on a scale. Scales belonging to a scale type share the same properties and the scale types are more or less powerful in the sense that more meaningful statements can be made the more powerful the scale is. The most commonly used scale types are described in the following subsection.

Measures can also be classified in two other ways: (1) if the measure is objective or subjective or (2) if the measure is direct or indirect. These classifications are further discussed in Sects. 3.4.3–3.4.4.

3.4.2 Scale Types

The most common scale types are the following[2] [38, 76, 77]:

Nominal The nominal scale is the least powerful of the scale types. It only maps the attribute of the entity into a name or symbol. This mapping can be seen as a classification of entities according to the attribute.
Possible transformations for nominal scales are those that preserve the fact that the entities can only be mapped one-to-one.
Examples of a nominal scale are classification, labeling, and defect typing.

Ordinal The ordinal scale ranks the entities after an ordering criterion, and is therefore more powerful than the nominal scale. Examples of ordering criteria are "greater than," "better than," and "more complex."

[2] Fenton et al. [76, 77] present a fifth scale type. This scale type is the absolute scale and is a special case of the ratio scale. The absolute scale is used when the value itself is the only meaningful transformation. An example of an absolute scale is counting.

The possible transformations for the ordinal scale are those that preserve the order of the entities, i.e., $M' = F(M)$ where M' and M are different measures on the same attribute, and F is a monotonic increasing function.
Examples of an ordinal scale are grades and software complexity.

Interval — The interval scale is used when the difference between two measures is meaningful, but not the value itself. This scale type orders the values in the same way as the ordinal scale but there is a notion of "relative distance" between two entities. The scale is therefore more powerful than the ordinal scale type.
Possible transformations with this scale type are those where the measures are a linear combination of each other, i.e., $M' = \alpha M + \beta$ where M' and M are different measures on the same attribute. Measures on this scale are uncommon in software engineering.
Examples of an interval scale are temperature measured in Celsius or Fahrenheit.

Ratio — If there exists a meaningful zero value and the ratio between two measures is meaningful, a ratio scale can be used.
Possible transformations are those that have the same zero and the scales only differ by a factor, i.e., $M' = \alpha M$ where M' and M are different measures on the same attribute.
Examples of a ratio scale are length, temperature measured in Kelvin, and duration of a software development phase.

The measurement scales are related to qualitative and quantitative research. Furthermore, they relate to which statistics can be used on the measures. This is further discussed in Chap. 11. According to Kachigan [124], qualitative research is concerned with measurement on the nominal and ordinal scales, and quantitative research treats measurement on the interval and ratio scales.

3.4.3 Objective and Subjective Measures

Sometimes the measurement of an attribute cannot be measured without considering the viewpoint it is taken from. We can divide measures into two classes:

Objective — An objective measure is a measure where there is no judgment in the measurement value and is therefore only dependent on the object that is being measured. An objective measure can be measured several times and by different researchers, and the same value can be obtained within the measurement error. Examples of objective measures are lines of code (LOC) and delivery date.

Subjective — A subjective measure is the opposite of the objective measure. The person making the measurement contributes by making some sort of judgment. The measure depends on both the object and the viewpoint

from which it is taken. A subjective measure can be different if the object is measured again. A subjective measure is mostly of nominal or ordinal scale type. Examples of subjective measures are personnel skill and usability.

Subjective measures are always subject to potential bias. This is further discussed in Sect. 3.4.6.

3.4.4 Direct or Indirect Measures

The attributes that we are interested in are sometimes not directly measurable. These measures must be derived through other measures that are directly measurable. To distinguish the direct measurable measures from derived measures, we divide the measures into direct and indirect measures.

Direct A direct measurement of an attribute is directly measurable and does not involve measurements on other attributes. Examples of direct measures are lines of code and the number of defects found in a test.

Indirect An indirect measurement involves the measurement of other attributes. The indirect measure is derived from the other measures. Examples of indirect measures are defect density (number of defects divided by the number of lines of code) and programmer's productivity (lines of code divided by the programmer's effort).

3.4.5 Measurements in Software Engineering

The objects that are of interest in software engineering can be divided into three different classes:

Process The process describes which activities are needed to produce the software.

Product The products are the artifacts, deliverables, or documents that result from a process activity.

Resources Resources are the objects, such as personnel, hardware, or software, needed for a process activity.

In each of the classes we also make a distinction between internal and external attributes [75]. An internal attribute is an attribute that can be measured purely in terms of the object. The external attributes can only be measured with respect to how the object relates to other objects. Examples of different software measures are shown in Table 3.2.

Often in software engineering, software engineers want to make statements on an external attribute of an object. Unfortunately, the external attributes are mostly indirect measures and must be derived from internal attributes of the object. The internal attributes are mostly direct measures.

Table 3.2 Examples of measures in software engineering

Class	Examples of objects	Type of attribute	Example of measures
Process	Testing	Internal	Effort
		External	Cost
Product	Code	Internal	Size
		External	Reliability
Resource	Personnel	Internal	Age
		External	Productivity

The measures are often part of a measurement program. Building software measurement programs is discussed by, for example, Grady and Caswell [94] and Hetzel [105].

Measurements in software engineering are different from measurements in other domains, for example, physics. In those domains, it is often clear what the attributes are and how they are measured. In software engineering, however, it is sometimes difficult to define an attribute in a measurable way with which everyone agrees [76]. Another difference is that it is difficult to prove that the measures are anything else but nominal or ordinal scale types in software engineering. Validation of the indirect measures is more difficult as both the direct measures and the models to derive the external measure have to be validated.

When conducting empirical studies, we are interested in the scale types of the measures as the statistical analysis depends on them. Formally, the statistical analysis methods depend upon the scale type, but the methods are mostly rather robust regarding scale type. The basic rule is that the more powerful scale types we use, the more powerful analysis methods we may use (see Chap. 11).

Many measures in software engineering are often measured with nominal or ordinal scales, or it is not proven that it is a more powerful scale type. This means that we cannot use the most powerful statistical analysis methods, which require interval or ratio scales, for the empirical studies we conduct.

Briand et al. [38] argue that we can use the more powerful statistical analysis even if we cannot prove that we have interval or ratio scales. Many of the more powerful statistical methods are robust to non-linear distortions of the interval scale if the distortions are not too extreme. If we take care and carefully consider the risks, we can make use of the more powerful statistical methods and get results that otherwise would be infeasible without a very large sample of measures.

3.4.6 Measurements in Practice

In practice metrics are defined by the researcher and then collected during the operation step of the empirical study. When it comes to how the metrics should be collected it is an advantage if it does not require too much effort by the subjects in the study. In many experiments subjects fill out forms to provide the data, but it is also

possible to define instrumenting systems where data is automatically collected, for example, by the development environment. Lethbridge et al. [157] discuss several general techniques for collection.

Since the collected metrics are the basis for the further analysis, the quality of the collected metrics is important for the continued analysis of the study. This means that it is important to really understand what kinds of metrics are collected and to be certain about what scale type they belong to. It is also important to understand what distribution they represent, in particular if they are normally distributed or not.

When it comes to the distribution, this could be investigated by descriptive statistics. The data can, for example, be plotted in a graph, or another technique for analyzing to what extent the data is normally distributed can be used. This is further elaborated in Chap. 11. When it comes to the scale type, this is based on how the metrics are defined and must be understood by the researcher when the metrics are defined.

How the metrics are defined can greatly affect how good they are in displaying what the researcher is interested in. For example, Kitchenham et al. [141] compare two ways of displaying productivity and show that a scatter plot displaying effort versus size gives better information than a chart showing productivity over time. A general piece of advice is to not use metrics that are constructed from the ratio of two independent measures unless one is sure to understand the measure's implication.

During the operation of the study it is essential to make sure that the collected data is correct. This means that the researcher should apply quality assurance procedures during the experiment, for example, by reviewing how subjects fill out forms, checking consistencies between different values, etc. Data validation is further discussed in Chap. 9.

A factor related to this concerns who is the inventor or owner of the aspects that are investigated in an experiment. Ideally someone other than the inventor of new methods should evaluate them in experiments and other research approaches, as recommended by Kitchenham et al. [137]. The inventor of a method naturally wants the method to perform well and there is always a risk that the researcher consciously or unconsciously selects metrics that are favorable for the investigated method. If it is known by the subjects that the researcher is the inventor of the investigated method this may also affect their performance. If experiments are carried out where own methods are studied, the design and selection of metrics could be reviewed by external researchers.

3.5 Empiricism to Improve

Why should we perform experiments and other empirical studies in software engineering? The major reasons for carrying out quantitative empirical studies is the opportunity of getting objective and statistically significant results regarding the understanding, controlling, prediction, and improvement of software development. Empirical studies are an important input to the decision-making in an improvement-seeking organization.

Before introducing new techniques, methods, or other ways of working, an empirical assessment of the virtues of such changes is preferred. A framework for evaluation of software process changes was presented in Sect. 2.7, where different empirical approaches were suggested in three different contexts: desktop, laboratory, and real-life projects.

To be successful in software development there are some basic requirements [16, 17, 57]:

1. Understanding of the software process and product
2. Definition of process and product qualities
3. Evaluation of successes and failures
4. Information feedback for project control
5. Learning from experience
6. Packaging and reuse of relevant experience

Empirical studies are important to support the achievement of these requirements, and fit into the context of industrial and academic software engineering research, as well as in a learning organization, seeking continuous improvement. An example of a learning organization, called Experience Factory, is proposed by Basili in conjunction with the Quality Improvement Paradigm [16], as further described in the next subsection. This approach also includes a mechanism for defining and evaluating a set of operational goals using measurement. This mechanism is called the Goal/Question/Metric (GQM) method [28], and is further described below. The GQM method is described in more detail by van Solingen and Berghout [257].

3.5.1 *Quality Improvement Paradigm*

The Quality Improvement Paradigm (QIP) [16] is a general improvement scheme tailored for the software business. QIP is similar to the Plan/Do/Study/Act cycle [33, 57], and includes six steps as illustrated in Fig. 3.2.

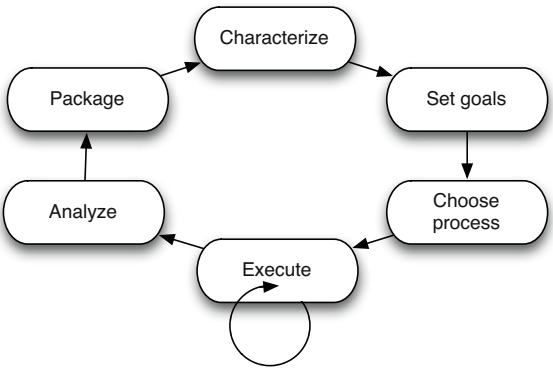


Fig. 3.2 The six steps of the Quality Improvement Paradigm [16]

These steps are explained below [29].

1. *Characterize*. Understand the environment based upon available models, data, intuition, etc. Establish baselines with the existing business processes in the organization and characterize their criticality.
2. *Set goals*. On the basis of the initial characterization and of the capabilities that have a strategic relevance to the organization, set quantifiable goals for successful project and organization performance and improvement. The reasonable expectations are defined based upon the baseline provided by the characterization step.
3. *Choose process*. On the basis of the characterization of the environment and the goals that have been set, choose the appropriate processes for improvement, and supporting methods and tools, making sure that they are consistent with the goals that have been set.
4. *Execute*. Perform the product development and provide project feedback based upon the data on goal achievements that are being collected.
5. *Analyze*. At the end of each specific project, analyze the data and the information gathered to evaluate the current practices, determine problems, record findings, and make recommendations for future project improvements.
6. *Package*. Consolidate the experience gained in the form of new, or updated and refined, models and other forms of structured knowledge gained from this and prior projects.

The QIP implements two feedback cycles [29] (see also Fig. 3.2):

- The *project feedback cycle* (*control cycle*) is the feedback provided to the project during the execution phase. Whatever the goals of the organization, the project used as a pilot should use its resources in the best possible way; therefore quantitative indicators at project and task level are useful to prevent and solve problems.
- The *corporate feedback cycle* (*capitalization cycle*) is the feedback loop that is provided to the organization. It has the double purpose of providing analytical information about project performance at project completion time by comparing the project data with the nominal range in the organization and analyzing concordance and discrepancy. Reusable experience is accumulated in a form that is useful and applicable to other projects.

3.5.2 Experience Factory

The QIP is based on the idea that the improvement of software development requires continuous learning. Experience should be packaged into experience models that can be effectively understood and modified. Such experience models are stored in a repository, called *experience base*. The models are accessible and can be modified for reuse in current projects.

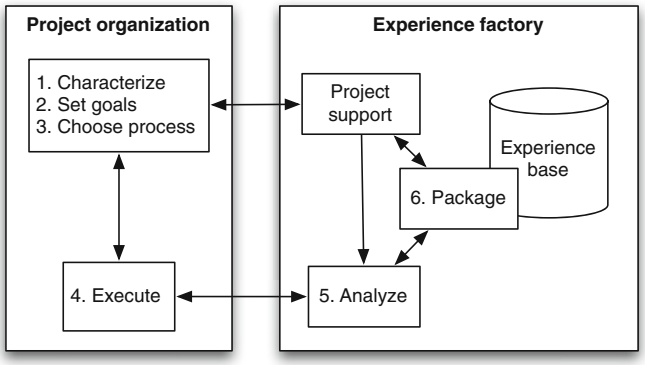


Fig. 3.3 Experience Factory

The QIP focuses on a logical separation of project development (performed by the Project Organization) from the systematic learning and packaging of reusable experience (performed by the Experience Factory) [17]. The Experience Factory is thus a separate organization that supports product development by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand (see Fig. 3.3).

The Experience Factory packages experience by "building informal, formal, or schematized models and measures of various processes, products, and other forms of knowledge via people, documents, and automated support" [29].

The goal of the Project Organization is to produce and maintain software. The project organization provides the Experience Factory with project and environment characteristics, development data, resource usage information, quality records, and process information. It also provides feedback on the actual performance of the models processed by the Experience Factory and utilized by the project.

The Experience Factory processes the information received from the development organization, and returns direct feedback to each project, together with goals and models tailored from similar projects. It also provides baselines, tools, lessons learned, and data, tailored to the specific project.

To be able to improve, a software developing organization needs to introduce new technology. It needs to experiment and record its experiences from development projects and eventually change the current development process. When the technology is substantially different from the current practice, the evaluation may be offline to reduce risks. The change evaluation, as discussed above, may take the form of a controlled experiment (for detailed evaluation in the small scale) or of a case study (to study the scale effects). In both cases, the Goal/Question/Metric method, as described subsequently, provides a useful framework.

3.5.3 Goal/Question/Metric Method

The Goal/Question/Metric (GQM) [28, 39, 257] method is based upon the assumption that for an organization to measure in a purposeful way it must:

1. Specify the goals for itself and its projects.
2. Trace those goals to the data that is intended to define those goals operationally.
3. Provide a framework for interpreting the data with respect to the stated goals.

The result of the application of the GQM method is a specification of a measurement model targeting a particular set of issues and a set of rules for the interpretation of the measurement data.

The resulting measurement model has three levels, as illustrated by the hierarchical structure in Fig. 3.4.

1. *Conceptual level* (Goal). A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Objects of measurement are products, processes, and resources (see also Sect. 3.4).
2. *Operational level* (Question). A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterization model. Questions try to characterize the objects of measurement (product, process, and resource) with respect to a selected quality aspect and to determine its quality from the selected viewpoint.
3. *Quantitative level* (Metric). A set of data is associated with every question to answer it in a quantitative way (either objectively or subjectively).

The process of setting goals is critical to the successful application of the GQM method. Goals are formulated based on (1) policies and strategies of the organization, (2) descriptions of processes and products, and (3) organization models. When goals have been formulated, questions are developed based on these goals. Once the questions have been developed, we proceed to associating the questions with appropriate metrics.

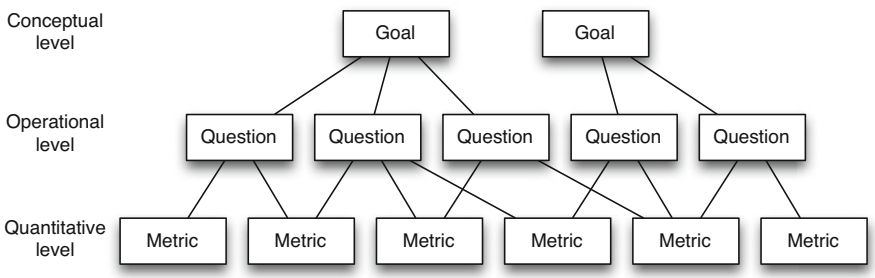


Fig. 3.4 GQM model hierarchical structure

Practical guidelines on how to use GQM for measurement-based process improvement are given by Briand et al. [39] and van Solingen and Berghout [257].

3.6 Empirically Based Technology Transfer

Empirical studies have a stand-alone value, but they can also be part of a knowledge exchange and improvement endeavor jointly between academia and industry, for example in technology transfer as also discussed above. Software engineering is an applied research area, and hence to perform research on industrially relevant problems is expected. It is in many cases insufficient to just do academic research on, for example, requirements engineering or software testing with the motivation that these areas are challenging in industry. Software engineering is preferably conducted jointly by academia and industry to enable transfer of knowledge in both directions and ultimately transfer of new methods, technologies, and tools from academia to industry. Joint research provides an excellent opportunity to improve industrial software development based on concrete evidence, and is hence a good example of evidence-based software engineering [63, 139].

Based on a long-term collaborative venture, a model for technology transfer was documented and presented by Gorschek et al. [92]. The seven steps in the model are summarized below to illustrate how different empirical studies and in particular experiments can be used in empirically driven improvement. The model is illustrated in Fig. 3.5. The model is closely related to the discussion about software process improvement in Sect. 3.5. The main focus of the model is on the usage of different empirical methods to create a solution to a real industrial problem and bring it to industrial application.

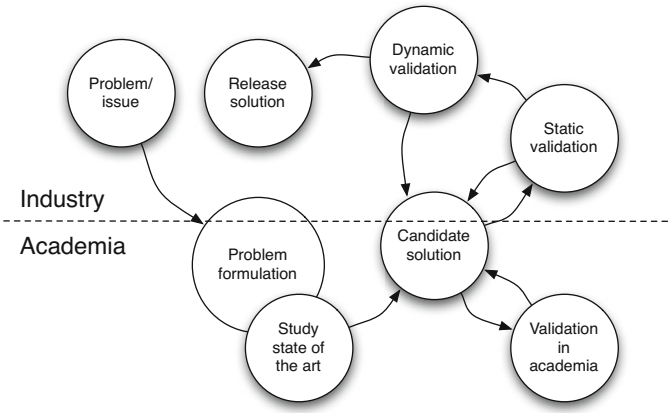


Fig. 3.5 Technology Transfer model. (Adapted from the description by Gorschek et al. [92])

Identification of Industrial Problem/Issue The first step is to identify actual industrial challenges in a specific industrial context, which implies that the researcher is present at the industrial partner(s). The identification of challenges may be done using, for example, a survey or interviews, which are briefly presented in Chap. 5. The objective is to capture the challenges and in particular issues that are suitable for research. It must be possible to formulate any challenge identified as a research problem to prevent the researcher from ending up in the role of a consultant addressing short-term problems.

A major benefit with doing this step thoroughly is that it creates an opportunity to build a joint trust and ensures that the industrial partner and its employees get used to having researchers present in their environment. At this stage, commitment from both management and other practitioners involved in the joint effort is crucial to ensure future success [275].

Problem Formulation Based on the challenge(s) identified, the challenge should be formulated as a research problem and research questions should be specified. If several different challenges are identified there is a need to prioritize which to address. Furthermore, a main contact person for the chosen challenge should be identified. The person should preferably not only be appointed; this should be a person who would like to be the driver within the company and act as a champion for the research collaboration. This includes helping to get in contact with the right people in the company, and helping to ensure that the researchers get access to systems, documentation, and data when needed.

As a natural part of the formulation of the research problem, the researchers conduct a literature search. This may be done as a systematic literature review as presented in Chap. 4. A literature review is needed to know about existing approaches to the identified industrial challenge. It provides a basis for understanding the relationship between approaches available and the actual industrial needs.

Candidate Solution Based on available approaches and the actual needs, a candidate solution is developed, which may include tailoring to the current processes, methods, technologies, and tools used at the company. The solution is preferably developed in close collaboration with the industrial partner(s) so that the applicability can be continuously ensured. Although a specific solution for a company may be derived, the intention of the researcher is to develop a generic solution, which is then instantiated in a specific context.

Validation in Academia A first validation of the proposed solution is preferably conducted in an academic environment to minimize the risk, i.e., an offline validation. In many cases this may be conducted as an experiment as described in several chapters of this book or as a small-scale evaluation in a student project. The validation in an academic environment may be conducted with either students as subjects or with representatives from the industrial partner(s).

The main objective in this step is to capture any obvious flaws in the proposed solution and to identify improvement proposals of the candidate solution. This is done in an academic setting to ensure that the best possible solution is available when bringing it to industry.

Static Validation In the static validation, industry representatives evaluate the candidate solution offline. This may be done through a presentation of the candidate solution followed by either interviews of different industry representatives, preferably in different affected roles, or joint workshops. In addition, it is preferable to make a general presentation to the organization to make them aware of the proposed solution at an early stage. This also gives an opportunity for the personnel to raise their voice at an early stage. It is hoped that this will help overcome any resistance once the new solution is going to be integrated into the way software is developed within the organization.

Based on the static validation, the new solution may need to be changed based on the feedback. The seven steps are all iterative, and hence it is more a matter of which order they start; they should definitively not be viewed as a waterfall approach without feedback cycles.

Dynamic Validation Once the new solution passes the static validation and there is agreement and commitment to implement the new solution, it is time to move to a dynamic validation. This is preferably done as a pilot evaluation. Exactly how to conduct the validation depends on the type of solution. The new solution may be used in a project, a subproject or for parts of a system, or for a specific activity. Independently, it is recommended that the dynamic validation be followed closely to evaluate the solution. The dynamic solution may be studied using a case study approach as described in Chap. 7.

Release Solution A generic solution must be tailored to each unique situation. There is a need to ensure that any research solution is properly handed over to an industrial champion and that the company has sufficient support in terms of descriptions, training, and potential tool support. The latter is not primarily the responsibility of the researchers, but they must support their collaborative partner(s) to ensure that the transfer of the new solution is properly in place and integrated into the organization before moving to the next industrial challenge.

Preferably, the broader usage is also studied empirically through a case study. This will help in obtaining empirical evidence for the new solution developed in the research collaboration.

Concluding Remark The transfer model outlined illustrates how different empirical approaches may be applied to support transfer of new research results from identification of needs to actual industrial usage.

Finally, it is interesting to note that the industrial representatives are primarily interested in the specific tailoring to their environment, while from a researcher's perspective it becomes a case for the generic solution. Thus, the collaborative partners may have different main focuses, but at the end they both benefit from the joint effort. The industrial partner gets a solution to an identified challenge and the researchers are able to evaluate a research result in a real industrial environment. Gorschek and Wohlin [91] present an example of a generic solution for requirements abstraction and a particular industrial instantiation of the approach is presented by Gorschek et al. [93] separately.

3.7 Exercises

3.1 Which are the key ethical principles to observe when conducting experiments?

3.2 Which role is played by replications and theories in building empirical knowledge?

3.3 What are measure, measurement, and metric and how do they relate?

3.4 Into which three classes are measurements in software engineering divided?

3.5 How can the Experience Factory be combined with the Goal/Question/Metrics method and empirical studies in a technology transfer context?

Chapter 4
Systematic Literature Studies

Systematic literature studies provide an essential foundation in an evidence-based research discipline. In 2004, Kitchenham et al. [139] suggested that software engineering would benefit from adopting an evidence-based approach. Since then systematic literature studies have become common in software engineering.

Systematic literature studies can be divided into *systematic literature reviews* and *systematic mapping studies*. Reviews are focused on synthesizing evidence in a defined area while systematic mapping studies aim at structuring an area. Systematic mapping studies are sometimes referred to as *scoping studies*. Systematic literature reviews, as a research method, are defined in Sect. 4.1. The different steps of systematic literature reviews are presented in Sects. 4.2–4.4. Mapping studies are discussed briefly in Sect. 4.5. Systematic literature studies may need to be updated as new research is published. This is discussed in Sect. 4.6. Section 4.7 concludes the chapter with a presentation of some examples of systematic literature reviews.

4.1 Definition of Systematic Literature Review

Systematic literature review is defined as follows:

Definition 4.1 A **systematic literature review** is a form of secondary study that uses a well-defined methodology to identify, analyze, and interpret all available evidence related to a specific research question in a way that is unbiased and (to a degree) repeatable [133].

Systematic literature reviews aim to give a complete, comprehensive, and valid picture of the existing evidence, and the identification, analysis, and interpretation must be conducted in a scientific and rigorous way. The objective is to identify and summarize all relevant studies for the review. These studies are referred to as primary studies. In order to conduct the review in a systematic way, Kitchenham

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_4

and Charters have adapted guidelines for systematic literature reviews to software engineering, primarily from medicine, evaluated them [36], and updated them accordingly [133]. Since then Kitchenham et al. [146] have provided a comprehensive presentation of systematic literature reviews and a detailed description of how systematic literature reviews ought to be reported [147]. Usman et al. [255] presented a quality assessment instrument for systematic literature reviews, which can be used by authors, reviewers, and readers to ensure the quality of systematic literature reviews.

The guidelines for systematic literature reviews consist of a three-step process for *planning, conducting*, and *reporting* the review. These are summarized below.

4.2 Planning the Review

Planning a systematic literature review includes several actions:

Identification of the Need for a Review The need for a systematic review originates from a researcher aiming to understand the state-of-the-art in an area, or from practitioners wanting to use empirical evidence in their strategic decision-making or improvement activities. If there are more or less systematic literature reviews available in the field, they should be appraised regarding scope and quality, to evaluate if they are sufficient to meet the current needs for a review. The appraisal can be done using the guidelines provided by Usman et al. [255]. The potential need to update a systematic literature study is further discussed in Sect. 4.6.

Specifying the Research Question(s) The area of the systematic literature review and the specific research questions set the focus for the identification of the primary studies, the extraction of data from the studies, and the analysis. Hence, the research questions must be well thought through and phrased. Aspects to take into account in phrasing the research questions include [133]:

- The *population* in which the evidence is collected, i.e., which group of people, programs, or businesses is of interest for the review.
- The *intervention* applied in the empirical study, i.e., which technology, tool, or procedure is under study.
- The *comparison* to which the intervention is compared, i.e., how is the control treatment defined? In particular the "placebo" intervention is critical, as "not using the intervention" is generally not a valid action in software engineering.
- The *outcomes* of the study should preferably be statistically significant, but should also be significant from a practical point of view. For example, it may be good to know that something is twice as time-consuming even it is 10% better in some respect. Although it will not result in a change of practice, its significance comes from it being a negative result.

- The *context* of the study must be defined, which is an extended view of the population, including whether it is conducted in academia or industry, in which industry segment, and also the incentives for the subjects [110, 190].
- The *research designs* to include in the research question must also be defined.

Staples and Niazi recommend the scope of a systematic literature review be limited by clear and narrow research questions to avoid unmanageable studies [241].

Developing a Review Protocol The review protocol defines the procedures for the systematic literature review. It also acts as a log for conducting the review. Hence, it is a "living" document that is of importance both for the practical conduct of the review and for its validity. Kitchenham and Charters propose the following items be covered in a review protocol [133]:

- Background and rationale
- Research questions
- Search strategy for primary studies
- Study selection criteria
- Study selection procedures
- Study quality assessment checklists and procedures
- Data extraction strategy
- Synthesis of the extracted data
- Dissemination strategy
- Project timetable

To ensure the quality of the review protocol, it is recommended to make an appraisal of the protocol before applying it, for example, by using the instrument presented by Usman et al. [255]. Moreover, the protocol is preferably reviewed by peers to ensure its consistency and validity. Experiences from conducting systematic literature reviews highlight the importance of a pre-review study to help in scoping the research questions, as well as being open to modifying research questions during the protocol development, as the problem under study becomes clearer [36].

4.3 Conducting the Review

Conducting the review means setting the review protocol into practice. This includes:

Identification of Research The main activity in this step involves specifying a search strategy. Two main search strategies exist. The first alternative is to identify search strings and decide which databases should be searched [133]. For this alternative it is essential to identify good search strings and to be able to adapt them to different databases since they are not necessarily constructed in the same way. Identifying some papers in advance that ought to be found in the search helps in validating and building trust in the search strings. The second alternative is to

search for primary studies based on references to and from other studies, which is called "snowballing" [211, 268]. A challenge when using snowballing is to identify a start set of primary studies on which to base the snowballing procedure. Guidelines for conducting snowballing in a software engineering context are presented by Wohlin [268]. A comparison of the two alternatives is presented by Jalali and Wohlin [118], and experiences from using both search strategies are discussed by Badampudi et al. [14]. A procedure called hybrid search strategy is proposed by Wohlin [268]. Mourão et al. [181] define and evaluate four alternative hybrid search strategies, which are further compared and evaluated by Mourão et al. [182] and Wohlin et al. [278] to mitigate the challenges with the two main alternative search strategies. The resulting suggestion is to perform a database search in Scopus to identify an appropriate start set of primary studies to start the snowballing procedure. Moreover, the two main search strategies, or the hybrid search strategy, can be complemented with manual searches in journals and conference proceedings, as well as searching researchers' Web sites or sending questions to researchers.

The search strategy is a trade-off between finding all relevant primary studies and not getting an overwhelming number of false positives, which must be excluded manually [58]. A false positive is an outcome that is positive when it should not be; in this case, it means that a paper is found and hence assumed to be of interest, and later it turns out that it is not and therefore it has to be removed. For the database search alternative, the search string should be developed from the area to be covered and the research questions. It is recommended to use multiple databases to try to cover all relevant literature, but it also creates duplicates, which must be identified and removed. The challenges are similar when using a snowballing search strategy. Ultimately, it must be accepted that the papers found are a sample of the population of all papers on a specific topic. Moreover, most often researchers limit the search for relevant literature to papers written in English, which in itself most likely implies that all relevant literature cannot be found. The key issue is that the sample is indeed from the intended population.

The published primary studies tend to have a *publication bias*, which means that (in some sense) *positive* results are more likely to be published than *negative* results. Hence, gray literature, like technical reports, theses, rejected publications, and work in progress, can also be considered [133]. Garousi et al. [85] provide guidelines for how to include gray literature in software engineering literature reviews. Furthermore, Kamei et al. [127] studied the use of gray literature in secondary studies in software engineering and acknowledge them for bringing practical perspectives into the study, although gray literature studies may be hard to find when looking for a primary study, or to trace if replicating a study, since they are not formally published in persistent storage.

The search results and a log of the actions taken should be stored, preferably using a reference management system.

Selection of Primary Studies The basis for the selection of primary studies is the inclusion and exclusion criteria. The criteria should be developed beforehand, to

avoid bias. However, they may have to be adjusted during the course of the selection, since all aspects of inclusion and exclusion are not apparent in the planning stage.

The identified set of candidate studies are processed related to the selection criteria. For some studies, it is sufficient to read the title or abstract to judge the paper, while other papers need a more thorough analysis of, for example, the methodology or conclusions to determine their status. Structured abstracts [42] may help the selection process.

As the selection process is a matter of judgments, also with well-defined selection criteria, it is advised that two or more researchers assess each paper, or at least a random sample of the papers. Then the inter-rater agreement may be measured using Cohen's Kappa coefficient [48] and be reported as a part of the quality assessment of the systematic literature review. However, it should be noted that a relatively high Cohen's Kappa coefficient may be obtained due to the fact that many papers found in the automatic search are easily excluded by the researchers when assessing them manually. Thus, it may be important to conduct the assessment in several steps, i.e., to start by removing those papers that are obviously not relevant although found in the search. Pérez et al. [189] provide further recommendations on how Cohen's Kappa coefficient can be used to guide systematic literature reviews.

Study Quality Assessment Assessing the quality of the primary studies is important, particularly when the studies report contradictory results. The quality of the primary studies may be used to analyze the cause of contradicting results or to weight the importance of individual studies when synthesizing results.

There is no universally agreed and applicable definition of "study quality." Attempts to map quality criteria from medicine did not map to the quality range of software engineering studies [62].

The most practically useful means for quality assessment are checklists, even though their empirical underpinning may be weak. A study by Kitchenham et al. also showed that at least three reviewers are needed to make a valid assessment [144]. Checklists used in quality assessment of empirical studies are available in the empirical software engineering literature [133, 144, 211, 255].

The quality assessment may lead to some primary studies being excluded if the study quality is part of the selection criteria. It is also worth noting that the quality of the primary studies should be assessed, not the quality of the reporting. However, it is often hard to judge the quality of a study if it is poorly reported. Contacts with authors may be needed to find or clarify information lacking in the reports.

Data Extraction and Monitoring Once the list of primary studies is decided, the data from the primary studies is extracted. A data extraction form is designed to collect the information needed from the primary study reports. If the quality assessment data is used for study selection, the extraction form is separated into two parts, one for quality data, which is filled out during quality assessment, and one for the study data to be filled out during data extraction.

The data extraction form is designed based on the research questions. For pure meta-analytical synthesis, the data is a set of numerical values, representing number of subjects, objects characteristics, treatment effects, confidence intervals, etc. For less homogeneous sets of studies, more qualitative descriptions of the primary studies must be included. In addition to the raw data, the name of the reviewer, date of data extraction, and publication details are logged for each primary study.

The data extraction form should be piloted before being applied to the full set of primary studies. If possible, the data extraction should be performed independently by two researchers, at least for a sample of the studies, in order to assess the quality of the extraction procedure.

If a primary study is published in more than one paper, for example, if a conference paper is extended to a journal version, only one instance should be counted as a primary study. Generally, the journal version is preferred, as it is most complete, but both versions may be used in the data extraction. Supporting technical reports or communication with authors may also serve as data sources for the extraction.

Data Synthesis The most advanced form of data synthesis is *meta-analysis*, although many systematic literature reviews use more qualitative approaches since it is infeasible to perform meta-analysis. Some qualitative approaches are presented below. Meta-analysis refers to statistical methods being applied to analyze the outcome of several independent studies. Meta-analysis assumes that the synthesized studies are homogeneous, or the cause of the in-homogeneity is well known [196]. A meta-analysis compares *effect sizes* and p values to assess the synthesized outcome. It is primarily applicable to replicated experiments, if any, due to the requirement on homogeneity. In summary, the studies to be included in a meta-analysis must [196]:

- Be of the same type, for example, formal experiments
- Have the same test hypothesis
- Have the same measures of the treatment and effect constructs
- Report the same explanatory factors

Meta-analysis procedures involve three main activities [196]:

1. Decide which studies to include in the meta-analysis.
2. Extract the effect size from the primary study report, or estimate if there is no effect size published.
3. Combine the effect sizes from the primary studies to estimate and test the combined effect.

In addition to the primary study selection procedures presented above, the meta-analysis should include an analysis of *publication bias*. Such methods include the *funnel plot*, as illustrated in Fig. 4.1 where observed effect sizes are plotted against a measure of study size, for example, the inverse of the variance or another dispersion measure (see Sect. 11.1.2). The data points should scatter around a “funnel” pattern

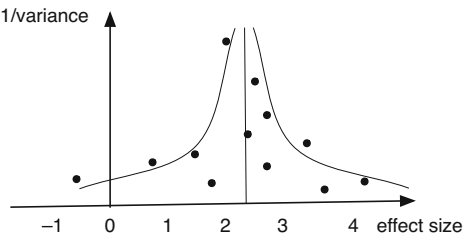


Fig. 4.1 An example funnel plot for 12 hypothetical studies

if the set of primary studies is complete. Gaps in the funnel indicate some studies not being published or found [196].

The *effect size* is an indicator, independent of the unit or scale that is used in each of the primary studies. It depends on the type of study, but could typically be the difference between the mean values of each treatment. This measure must be normalized to allow for comparisons with other scales, that is, divided by the combined standard deviation [196].

The analysis assumes homogeneity between studies, and is then done with a *fixed effects* model. The meta-analysis estimates the true effect size by calculating an average value of the individual study effect sizes, which are averages themselves. There are tests to identify heterogeneity, such as the Q test and the Likelihood Ratio test, which should be applied to ensure model conditions are met [196].

For heterogeneous data, there is a *random effects* model, which allows for variability due to an unknown factor that influences the effect sizes for the primary studies. This model provides estimates both for the sampling error, as the fixed effects model, and for the variability in the heterogeneous sub-populations.

Less formal methods for data synthesis include *descriptive* and *narrative synthesis*. These methods tabulate data from the primary studies in a manner that brings light to the research question. As a minimum requirement on tabulated data, Kitchenham and Charters propose the following items be presented [133]:

- Sample size for each intervention
- Estimates of effect size for each intervention with standard errors for each effect
- Difference between the mean values for each intervention, and the confidence interval for the difference
- Units used for measuring the effect

Statistical results may be visualized using *forest plots*. A forest plot presents the means and variance of the difference between treatments for each study. An example forest plot is shown in Fig. 4.2.

Synthesizing heterogeneous studies and mixed-methods studies often requires qualitative synthesis approaches. Cruzes and Dybå [53] surveyed secondary studies in software engineering, which included synthesis of empirical evidence. They identified six qualitative synthesis methods used in software engineering. These methods are briefly introduced below. For more detail, refer to Cruzes and Dybå [53, 54] and related references.

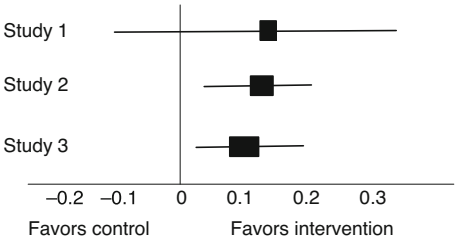


Fig. 4.2 An example forest plot for three hypothetical studies

- *Thematic analysis* is a method aiming at identifying, analyzing, and reporting patterns or themes in the primary studies. At minimum, it organizes and presents the data in rich detail, and interprets various aspects of the topic under study.
- *Narrative synthesis*, mentioned above, tells a "story" which originates from the primary evidence. Raw evidence and interpretations are structured, using, for example, tabulation of data, groupings and clustering, or vote-counting as a descriptive tool. Narrative synthesis may be applied to studies with qualitative or quantitative data, or combinations thereof.
- The *comparative analysis* method is aimed at analyzing complex causal connections. It uses Boolean logic to explain relations between cause and effect in the primary studies. The analysis lists necessary and sufficient conditions in each of the primary studies and draws conclusions from presence/absence of independent variables in each of the studies. This is similar to Noblit and Hare's [186] *Line of argument synthesis*, referred to by Kitchenham and Charters [133].
- The *case survey* method is originally defined for case studies, but may apply to heterogeneous experiments too. It aggregates existing research by applying a survey instrument of specific questions to each primary study [161], similar to the data extraction mentioned above. The data from the survey is quantitative, and hence the aggregation is performed using statistical methods [153].
- *Meta-ethnography* translates studies into one another and synthesizes the translations into concepts that go beyond individual studies. Interpretations and explanations in the primary studies are treated as data in the meta-ethnography study. This is similar to Noblit and Hare's [186] *Reciprocal translation* and *Refutational synthesis*, referred to by Kitchenham and Charters [133].
- *Scoping analysis* aims at giving an overview of the research in a field, rather than synthesizing the findings from the research. Scoping is also referred to as mapping studies, which are further discussed in Sect. 4.5.

Independently of the synthesis method, a *sensitivity analysis* should take place to analyze whether the results are consistent across different subsets of studies. Subsets of studies may be, for example, high-quality primary studies only, primary studies of a particular type, or primary studies with good reports, presenting all details needed.

4.4 Reporting the Review

Like any other empirical study, the systematic literature review may be reported to different audiences. In particular, if the purpose of the review is to influence practitioners, the format of the report has to be tailored to its audience. Kitchenham and Charters [133] list the following forms for dissemination targeting practitioners:

1. Practitioner-oriented journals and magazines
2. Press Releases to the popular and specialist press
3. Short summary leaflets
4. Posters
5. Web pages
6. Direct communication to affected bodies

For academic audiences, the detailed reporting of procedures for the study is critical for the ability to assess and evaluate the quality of the systematic literature review. The reporting ideally includes changes to the study protocol, complete lists of included and excluded primary studies, data on their classification, as well as the raw data derived from each of the primary studies. If space constraints do not allow all details to be published, it is recommended that a supporting technical report be published online. A detailed structure for the academic report is proposed by Kitchenham and Charters [133], and further developed and presented by Kitchenham et al. [147].

4.5 Mapping Studies

If the research question for the literature study is broader, or the field of study is less explored, a *mapping study* may be launched instead of a systematic literature review. A mapping study [192], sometimes referred to as *scoping study* [133], searches a broader field for any kind of research to get an overview of the state-of-the-art or state-of-practice on a topic. Guidelines for conducting mapping studies are presented by Petersen et al. [192, 193].

A mapping study follows the same principal process as for systematic literature reviews, but has different criteria for inclusions/exclusions and quality. Due to its broader scope and varying types of studies, the collected data and the synthesis tend to be more qualitative than for systematic literature reviews. However, it is important for the contribution and relevance of a mapping study that the analysis goes beyond the pure descriptive statistics and relates the trends and observations to real-world needs.

Kitchenham et al. [145] provided a summary of the key characteristics for mapping studies compared to systematic literature reviews, which is presented in Table 4.1.

Table 4.1 Difference between mapping studies and systematic literature reviews, according to Kitchenham et al. [145]

SLR elements	Systemic mapping study	Systematic literature review
Goals	Classification and thematic analysis of literature on a software engineering topic	Identifying best practice with respect to specific procedures, technologies, methods, or tools by aggregating information from comparative studies
Research question	Generic – related to research trends. Of the form: which researchers, how much activity, what type of studies, etc.	Specific – related to outcomes of empirical studies. Of the form: Is technology/method A better or not than B?
Search process	Defined by topic area	Defined by research question which identifies the specific technologies being investigated
Scope	Broad – all papers related to a topic area are included but only classification data about these are collected	Focused – only empirical papers related to a specific research question are included and detailed information about individual research outcomes is extracted from each paper
Search strategy requirements	Often less stringent if only research trends are of interest; for example, authors may search only a targeted set of publications, restrict themselves to journal papers, or restrict themselves to one or two digital libraries	Extremely stringent – all relevant studies should be found. Usually systematic literature review teams need to use techniques other than simply searching data sources, such as looking at the references in identified primary studies and/or approaching researchers in the field to find out whether they are undertaking new research in the area
Quality evaluation	Not essential. Also complicated by the inclusive nature of the search, which can include theoretical studies as well as empirical studies of all types, making the quality evaluation of primary studies complicated	Important to ensure that results are based on best-quality evidence
Results	A set of papers related to a topic area categorized in a variety of dimensions and counts of the number of papers in various categories	The outcomes of the primary studies are aggregated to answer the specific research question(s), possibly with qualifiers (e.g., results apply to novices only)

4.6 Updating Systematic Literature Studies

Software literature studies become outdated. Mendes et al. [168] conclude that more than 430 systematic literature reviews were published in software engineering between 2004 and 2016. However, they identified only 20 updated systematic literature reviews, which suggests that many reviews are potentially outdated. As new primary studies emerge, further evidence becomes available which may or may not affect the previous conclusions. Thus, as part of an evidence-based discipline there is a need to keep systematic literature studies up to date.

Garner et al. [84] propose a definition of what constitutes an update. An update typically includes new primary studies which may include changes concerning data, analyses, or methods in relation to the previous version of the systematic literature review. To decide whether or not a systematic literature study needs to be updated, Mendes et al. [168] put forward a framework for assessing the need for an update. They applied the framework to the 20 systematic literature updates identified, and concluded that only six of them needed to be updated. Thus, updating outdated systematic literature studies is essential, but only studies needing an update should be updated. To support updating systematic literature studies, Wohlin et al. [277] present a search strategy for updating systematic literature studies.

4.7 Evolution of Systematic Literature Studies

As systematic literature studies have become more and more common, tertiary studies are also published. A tertiary study is a study of systematic literature studies. The tertiary studies are often targeting systematic literature studies in a specific area with the objective to summarize the evidence presented in the respective secondary studies. Kitchenham et al. [142, 143] published one of the first tertiary studies. They report 53 unique systematic literature reviews in software engineering being published between 2004 and 2008. It is concluded that there is a growth in the number of systematic literature reviews being published, and that the quality of the reviews tends to be increasing too. However, still there is large variation between those who are aware of and use any systematic guidelines for its conduct, and those who are not referring to any guidelines. Since then, several hundred systematic literature studies have been published [168], and tertiary studies have become more common too.

Since the tertiary study by Kitchenham et al. [142, 143], other tertiary studies have been published. They include both studies directed toward the actual conduct of systematic literature studies and studies targeting a specific area in software engineering. For example, Yang et al. [280] investigated how the quality of primary studies was assessed in systematic literature reviews. They conclude that quality assessment of primary studies has improved over the years. Imtiaz et al. [116] present a study on experiences from conducting systematic literature

reviews and Neto et al. [185] provide a study on multivocal literature reviews in software engineering. Ampatzoglou et al. [3] performed a tertiary study in software engineering with a focus on threats to validity, and proposed an analysis schema, comprising of study selection validity, data validity, and research validity. Tertiary studies targeting specific areas within software engineering include, for example, a tertiary study by Hoda et al. [108] on agile software development, a study by García-Mireles et al. [83] on gamification in software engineering, and a study by Kotti et al. [150] on machine learning in software engineering.

In the early days of systematic literature reviews in software engineering, some reviews focused on the type of studies conducted and different aspects of software engineering research.

For example, Sjøberg et al. [234] survey the experimental studies conducted in software engineering. They searched nine journals and three conference proceedings in the decade from 1993 to 2002, scanning through 5,453 articles to identify 103 experiments, i.e., 1.9% of the papers presented experiments. The two most frequent research categories are Software life cycle/engineering (49%) and Methods/Techniques (32%) classified according to Glass et al.'s scheme [89]. This finding is due to the relatively large number of experiments on inspection techniques and object-oriented design techniques in the investigated time span.

Using the same set of primary studies, Dybå et al. [64] reviewed the statistical power in software engineering experiments, and Hannay et al. [100] reviewed the use of theory in software engineering. Dieste et al. [58] investigated different search strategies on the same set of studies, whether titles, abstracts, or full texts should be searched, and also aspects related to which databases to search.

Early attempts at synthesizing five experiments on inspection techniques by Hayes [104] and Miller [173] indicate that the software engineering experiments in this field are not sufficiently homogeneous to allow for application of statistical meta-analysis. They also conclude that raw data must be made available for meta-analysts, as well as additional non-published information from the primary study authors.

However, the conduct of systematic literature studies has continued to be a concern. For example, van Dinter et al. [256] present a study on automation for conducting systematic literature reviews and conclude that most automation efforts are directed toward the selection of primary studies while limited support is provided for other steps in the process. Ros et al. [208] propose a machine learning approach to semi-automated search and selection in systematic literature studies.

Systematic literature studies also follow the trends in software engineering research and practice. For example, Hannay et al. [101] performed a systematic literature review on the effectiveness of pair programming; they conducted meta-analysis on data from 18 primary studies. They report separate analyses for three outcome constructs: quality, duration, and effort. They also visualize the outcomes using forest plots.

As new areas related to software engineering emerge, systematic literature studies on these topics are published. This can be exemplified with systematic

literature in deep learning in software engineering [279, 282] and cybersecurity in software engineering [126].

4.8 Exercises

4.1 What are the difference between a systematic literature review and a more general literature review?

4.2 What search strategies exist for primary studies?

4.3 Why should at least two researchers conduct some of the activities in a systematic literature study?

4.4 What is required of a primary study to be included in a meta-analysis?

4.5 Which are the key differences between a systematic literature study and a mapping study?

Chapter 5
Surveys

In general language a survey provides an overview of something, be it a geographical area, a subject, or opinions. Surveys are typically used for opinion polls and market research. In a scientific context, it has a more precise meaning. Fink [78] defines a survey as follows:

Definition 5.1 A **survey** is a system for collecting information from or about people to describe, compare, or explain their knowledge, attitudes, and behavior.

A survey is often an investigation performed in retrospect of, for example, a tool or technique, when it has been in use for a while [194], or potentially before it is used. It could be seen as a snapshot of the situation to capture the current status. The primary means of gathering qualitative or quantitative data are interviews or questionnaires. However, if the survey is conducted using interviews, it is preferable to label it as an interview study since it makes it clear upfront that it is a study using interviews. Both questionnaire-based surveys and interviews are done through taking a sample which is intended to be representative of the target population. The results from the study are analyzed to derive descriptive and explanatory conclusions. They are then generalized to the population from which the sample was taken. Storey et al. [247] call these studies "sample surveys" to distinguish them from "judgement studies," where experts are asked in their capacity of having knowledge on the topic, rather than being a sample from a population. Surveys are discussed further by, for example, Fink [78] and Robson [207].

When performing survey research the interest may be, for example, in studying how a new development process has improved the developer's attitudes toward quality assurance or prioritizing quality attributes [130]. Thus, a sample of developers is selected from all the developers at the company. A questionnaire is constructed to obtain information needed for the research. The questionnaire is answered by the sample of developers. The information collected is arranged into a form that can be handled in a quantitative or qualitative manner.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_5

The description below of the survey objective and design is inspired by the technical report by Linåker et al. [158]. Other guidelines for survey research have been presented by Kitchenham and Pfleeger [134] and Kasunic [131]. Molléri et al. [179] provide an annotated overview of guidelines for survey research in software engineering, while Ghazi and Petersen [86] discuss problems and mitigation strategies for survey research. Stavru addresses trustworthiness of surveys, through a review of example surveys on agile software engineering practices [244], and Baltes and Ralph [15] provide a critical review and guidelines for sampling in software engineering research.

5.1 Survey Characteristics

Surveys are almost never conducted to create an understanding of the particular sample. Instead, the purpose is to understand the population from which the sample was drawn [13]. For example, by inviting 25 developers to respond to a questionnaire on what they think about a new process, the opinion of the larger population of 100 developers in the company can be assessed. Surveys aim at development of generalized conclusions.

Surveys have the ability to provide a large number of variables to evaluate, but it is necessary to aim at obtaining the greatest degree of understanding from the smallest number of variables since this reduction also eases the data collection and analysis work. Surveys with many questions are tedious for respondents to fill out, and the data quality may consequently decline. On the other hand, surveys aim at providing broad overviews, which may require a broad range of questions.

The two most common means for data collection are questionnaires and interviews [78]. Questionnaires could be provided in paper form or in some electronic form, for example, via email or a Web-based questionnaire. The basic method for data collection through questionnaires is to send out the questionnaire together with instructions on how to fill it out, or alternatively to provide the information online and invite participants by sending them a link. The responding person answers the questionnaire and returns it to the researcher.

Interviews (by face-to-face or digital media) have some advantages in comparison with questionnaires:

- Interview surveys typically achieve higher response rates than, for example, email or Web-based surveys.
- An interviewer generally decreases the number of “do not know” and “no answer,” because the interviewer can answer questions about the questionnaire.
- It is possible for the interviewer to observe and ask questions.

The main disadvantage with interviews is the cost and time, which depend on the size of the sample, and it is also related to the intentions of the investigation, including the accessibility to the participants. If it is a survey within a company to which the researcher has access then interviews are preferable, but if the

population is spread geographically, potentially speaking different languages, then a questionnaire may be easier.

5.2 Survey Purposes

Surveys are divided into the following three types, i.e., their general objective is to do one of the following [13]:

- Descriptive
- Explanatory
- Explorative

Descriptive surveys can be conducted to enable assertions about a population. This could be determining the distribution of certain characteristics or attributes. The concern is not about why the observed distribution exists, but instead what that distribution is.

Explanatory surveys aim at making explanatory claims about the population. For example, when studying how developers use a certain testing technique, we might want to explain why some developers prefer one technique while others prefer another. By examining the relationships between different candidate techniques and several explanatory variables, we may try to explain why developers choose one of the techniques.

Finally, explorative surveys are used as a pre-study to a more thorough investigation to ensure that important issues are not missed. An example could be creating a loosely structured questionnaire, often including qualitative elements, and letting a sample from the population answer it. The information is gathered and analyzed, and the results are used to improve the full investigation. In other words, the explorative survey does not answer the basic research question, but it may provide new possibilities that could be analyzed and should therefore be followed up in the more focused or thorough survey. This relates to evaluation of the questionnaire and its questions before it is used for the actual study. This is further discussed in Sect. 5.4.

5.3 Survey Research Objective

The motivation for the research and the overall research questions should be determined as a starting point for going through the decision-making structure in Chap. 2. Assuming that a survey was an appropriate research method for addressing the overall research question, it is suitable to refine the overall research question into more specific questions to answer through the survey. The breakdown from an overall research question to more specific questions can be done using GQM (see Sect. 3.5.3). Using GQM also allows the researcher to identify which measures

should be a result from the survey. Kitchenham and Pfleeger [135] identify three main common type of objective in software engineering:

- *Frequency:* We may be interested in how often certain events, situations, or outcomes happen.
- *Severity:* When something happens, we may want to find out how severe it is, for example, in terms of overruns, quality, or some other essential characteristic of the project or product.
- *Influencing factors:* In other instances, we might like to know what factors influence a specific characteristic in software development.

Surveys come with challenges concerning participants and response rates. It is essential to have both a sufficient number of participants and as high a response rate as possible. Thus, there is a need to have clear boundaries of the objective, i.e., to decide on what should be included in the survey and what should not. It can be tempting to include too many aspects in a survey. However, most likely the response rate will suffer. Thus, the boundaries of the survey are crucial.

The main focus here is on self-administered questionnaires, but most issues to consider in a survey using questionnaires are also relevant in an interview study. The resources, such as people and tools, needed to perform the survey should be determined. People include fellow researchers and participants. More than one researcher should be involved in creating the questionnaire to ensure that the questionnaire is of suitable length and that the questions are appropriate for the research. Preferably, one or more independent reviewers of the questionnaire are identified and their commitment should be ensured. It is necessary to make sure that the questions are well formulated to minimize the risk of having questions that will be hard for the participants to understand. There is also a need to consider if it will be viable to recruit the appropriate participants given the research objective, and how they should be approached. Moreover, the researchers should consider how the questionnaire should be distributed. The most common approach is to use a survey tool online and invite participants to respond to it via the Web. However, several tools exist and it has to be decided which survey tool to use. Selection criteria include the survey features offered, cost of tool usage, as well as integrity and data protection for the respondents' potentially sensitive data.

The amount of work should be considered both for the researchers and the participants. Questionnaires with a large number of questions should be avoided since it will inevitably affect the response rate. It gets even worse if the questions have long and complex formulations.

When considering participants, three categories should be considered. First, the researchers should consider which *population* the questionnaire is targeting. It is essential to be clear about the population since the objective is to draw conclusions for the population. The formulation of the population needs to be aligned with the participants that can potentially be recruited. Second, a *sample frame* should be determined. A sample frame is a list of people from which the sample of participants will be recruited. Finally, it should be decided how to *sample* from the sample frame. The intention is that the sample frame should be a representative subset of the

population, and that the sample should be a subset of the sample frame. However, it is not without challenges.

To illustrate the concepts of population, sample frame and sample, and potential challenges, consider the following example. If the population is intended to be software engineers in Sweden, then we may decide that the sample frame is persons on LinkedIn with the work title "software engineer" located in Sweden, and that we randomly select persons to invite from those in the sample frame. In this case the intention is that the conclusions should be representative of software engineers in Sweden. However, we need to consider if people on LinkedIn are representative of the intended population, otherwise there is a potential risk for bias. Furthermore, software developers on LinkedIn may have chosen to use a different title than "software engineer." Thus, although it may initially seem like we will get a representative sample of the intended population, we do not know if either the sample frame or the sample is the subset as intended. Moreover, it is important to note that it does not mean that the findings can be generalized to software engineers in other countries.

The final activity in this step is to consider and decide how the findings will be used. This includes how and where to submit a publication, feedback to the participants, and if the intention is to try to influence practice based on the findings and if so, how.

5.4 Survey Design

A survey is a "fixed design" [207] study. This means that once the questionnaire has been designed and sent to the sample in our survey, it is too late to make any changes to the questionnaire. Thus, the design, review, and evaluation are important activities for survey research.

When designing the questionnaire, it is crucial to be selective of what to include and not. Furthermore, the questionnaire should be self-contained, i.e., preferably no additional documentation should be needed to understand the questions in the questionnaire. It may be essential to characterize the participants in the survey. If we assume that we perform a survey of some aspect of software development in industry, it may be important to, for example, ask about the respondent's role in the company, years of experience in the role, and the size of the company. Thus, the participants need to be characterized, i.e., descriptive data such as demographic information should be collected. Moreover, the questionnaire should be focused on the most essential questions to be able to respond to the research question. The questions may concern behavioral and attitudinal data. Behavioral questions are concerned with capturing changes in the behavior of the respondents. Attitudinal questions relate to opinions of the respondents.

The questions need to be carefully formulated and it should be decided how the outcome from the survey will be analyzed. This relates to how the questions are posed. We may decide to have open-ended questions, which means that we must use

qualitative methods to analyze the responses. Alternatively, we may have closed-end questions, in which case quantitative analysis methods could be used. The type of analyses depends on the type of data collected, as discussed in Sect. 3.4.2. The type of data also relates to which analysis could be done. Different types of analysis for experiments are presented in Chap. 11, but the analysis techniques are also useful for quantitative survey data. Likert scale data is often collected in surveys, which is data with a given number of alternatives, typically with a neutral value in the middle and with values measuring attitudes or opinions from one extreme to the opposite extreme, for example, strongly disagree to strongly agree. Five or seven alternatives are common when using the Likert scale. However, this type of data requires particular care in the analysis [46].

Once the questionnaire is designed, it should be reviewed. Feedback on the content and formulations is crucial to minimize the risk when the questionnaire is sent to the sample of persons targeted for the survey. After a review, it is recommended to also run a pilot of the questionnaire. This means identifying a few individuals who would be potential participants in the survey and could respond to the questionnaire and provide reflections on the content and the formulations to further improve the questionnaire. If no major changes are needed, the responses could, with some care, potentially be included in the analysis of the survey.

Next, based on the chosen population and sample frame, it should be decided how the sampling should be conducted to obtain our sample. We have two main types of sampling, i.e., probability sampling and non-probability sampling. Several different sampling approaches within the two types of sampling exist. Baltes and Ralph provide a primer on sampling methods and a critical review of survey research in software engineering [15]. Berndt [34] presents a summary of sampling methods for survey research. Ideally, from a theoretical point of view, complete random sampling is the best option. However, the two most commonly used sampling methods in software engineering are purposive sampling and convenience sampling [15], which are both non-probability sampling methods. In purposive sampling, the most useful subjects or items are selected. Convenience sampling leads to validity concerns since the participants are chosen based on availability and willingness to participate, which may substantially affect the conclusions from the survey.

Before distributing the questionnaire, validity threats should be considered, and when possible mitigated. The validity threats for empirical studies of different types are often similar, although some threats may be more common for some types of empirical studies than others. Validity evaluation and validity threats to consider for experiments are presented in Sects. 9.7 and 9.8, respectively. These sections form a good starting point for surveys too. However, there is a need to consider if some specific validity concerns exist for your survey that are not covered by the validity threats presented in relation to experiments.

5.5 Concluding Remarks

The challenges with conducting survey research in software engineering should not be underestimated. Ghazi and Petersen [86] highlight concerns in survey research, but also mitigation strategies. Sampling is one of the main concerns in survey research. Baltes and Ralph [15] provide a primer on sampling methods, and Berndt [34] presents a summary of sampling methods. Stavru [244] discusses thoroughness in survey research in software engineering based on a critical examination of industrial surveys concerning the use of agile methods, including issues related to population and sampling. The examination was conducted using four criteria for trustworthiness by Guba [97]. Guba's four criteria for trustworthiness are truth value, applicability, consistency, and neutrality. Based on the examination, Stavru [244] presents 21 criteria for thoroughness for reporting surveys. It is argued that the trustworthiness of a survey is reflected in the thoroughness of the reporting.

5.6 Exercises

5.1 What is a survey? Give examples of different types of surveys in software engineering.

5.2 Which are the three types of surveys?

5.3 When conducting a survey, the specific objective can be divided into the following three: frequency, severity, and influencing factors. Explain the differences between them in your own words.

5.4 Which aspects are essential to consider when designing a survey?

5.5 Why is reviewing and piloting surveys essential?

Chapter 6
Experiments

Experiments are the main focus of this book and we define *experiment* as follows:

Definition 6.1 An **experiment** (or controlled experiment) in software engineering is an empirical enquiry that manipulates one factor or variable of the studied setting. Based on randomization, different treatments are applied to or by different subjects, while keeping other variables constant, and measuring the effects on the outcome variables. In human-oriented experiments, humans apply different treatments to objects, while in technology-oriented experiments, different technical treatments are applied to objects.

Experiments are mostly done in a laboratory environment, which provides a high level of control. When experimenting, subjects are assigned to different treatments at random. The objective is to manipulate one or more variables and control all other variables at fixed levels. The effect of the manipulation is measured, and based on this a statistical analysis can be performed. In cases where it is impossible to randomly assign treatments to subjects, we may use quasi-experiments. Quasi-experiment is defined as follows:

Definition 6.2 **Quasi-experiment** is an empirical enquiry similar to an experiment, where the assignment of treatments to subjects cannot be based on randomization, but emerges from the characteristics of the subjects or objects themselves.

Experimental methods occasionally appear under other labels. For example, *A/B testing* is an experimental method, although not explicitly labeled as an experiment. A/B testing has been around for a long time and it could be viewed as a shorthand for comparing two alternatives to decide which is the best. It is the simplest form of a controlled experiment.

Nowadays, A/B testing is used in operational decision-making when evaluating different alternatives in an industrial setting. It can, for example, be used to assess user experience with a controlled experiment, such as comparing two Web page designs or to compare different versions of a feature. In general, A/B testing is used

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_6

by companies to compare different solutions. It is done in cycles, which is called continuous experimentation [79]. Continuous experimentation and A/B testing are established industry practices [10]. The experimental process is the same for A/B testing as for other controlled experiments, although integrated with the software engineering process and conducted with live users who are randomly assigned to the different versions [72].

Experiments can also be conducted in *simulation research*, based on real data, being mined from software repositories. The research involves applying an intervention to the data, controlling its outcome under strict control of factors. However, a significant share of studies labeled as experiments are not properly categorized, as observed by Ayala et al. [12], who analyzed 254 mining studies, published between 2015 and 2018, labeled as experiments, and found only 19% to be formal randomized controlled experiments.

Experiments are appropriate to investigate different aspects [100, 236], including:

- Confirm theories, i.e., to test existing theories
- Confirm conventional wisdom, i.e., to test people's conceptions
- Explore relationships, i.e., to test that a certain relationship holds
- Compare alternatives, i.e., to determine whether one alternative is better than another one
- Evaluate the accuracy of models, i.e., to test that the accuracy of certain models is as expected
- Validate measures, i.e., to ensure that a measure actually measures what it is supposed to

The strength of an experiment is that it can investigate in which situations the claims are true and can provide a context in which certain standards, methods, and tools are recommended for use.

Experiments are launched when we want control over the situation under study and want to manipulate behavior directly, precisely, and systematically. Also, experiments involve more than one treatment and are set up to compare the outcomes from the different treatments. For example, if it is possible to control who is using one method and who is using another method, and when and where they are used, it is possible to perform an experiment. This type of manipulation can be made in an offline situation, for example in a laboratory under controlled conditions, where the events are organized to simulate their appearance in the real world. Experiments may alternatively be made in a real-life context, which means that the investigation is executed in the field [13]. A/B testing is an example of conducting the study, or experiment, in a real-life setting. The level of control is more difficult in a real-life situation, but it may be possible to control some factors while it may not be possible to control others.

Experiments may be *human-oriented* or *technology-oriented*. In human-oriented experiments, humans apply different treatments to objects; for example, two inspection methods are applied to two pieces of code. In technology-oriented experiments, typically different tools are applied to objects, for example, two test case generation

tools are applied to the same programs. The human-oriented experiment has less control than the technology-oriented one, since humans behave differently on different occasions, while tools are (mostly) deterministic. Furthermore, due to learning effects, a human subject cannot apply two methods to the same piece of code, which two tools can do without bias.

The notion of context is essential in empirical studies in software engineering. Examples of different contexts could be the application area and the system type [190]. In an experiment, we identify the contexts of interest and their variables and sample over them. This means that we select objects representing a variety of characteristics that are typical for the organization in which the experiment is conducted and design the research so that more than one value will be measured for each characteristic. An example could be to investigate the effect of an inspection (or review) method with respect to the faults found in tests in two different systems, using two different programming languages, for example, in a situation where an organization has moved from one programming language to another. Then the different systems are the context for evaluating the inspection method, and hence similar objects are needed in the experiment. The inspection method becomes the independent variable and an experiment will involve objects where the different programming languages are used.

The experiment should be designed so that the objects involved represent all the methods we are interested in. Also, it is possible to consider the current situation to be the baseline (control), which means that the baseline represents one level (or value) of the independent variable, and the new situation is the one we want to evaluate. Then the level of the independent variable for the new situation describes how the evaluated situation differs from the control. However, the values of all the other variables should stay the same, for example, application domain and programming environment.

6.1 Experiment Principles

One of the main advantages of an experiment is the control of, for example, subjects, objects, and instrumentation. This ensures that we are able to draw more general conclusions. Other advantages include ability to perform statistical analysis using hypothesis testing methods and opportunities for replication. To ensure that we make use of the advantages, we need a process supporting us in our objectives in doing experiments correctly (the notion of experiments includes quasi-experiments, unless clearly stated otherwise). The basic principles behind an experiment are illustrated in Fig. 6.1.

The starting point is that we have an idea of a cause and effect relationship, i.e., we believe that there is a relationship between a cause construct and an effect construct. We have a theory or are able to formulate a hypothesis. A hypothesis means that we have an idea of, for example, a relationship, which we are able to state formally in a hypothesis.

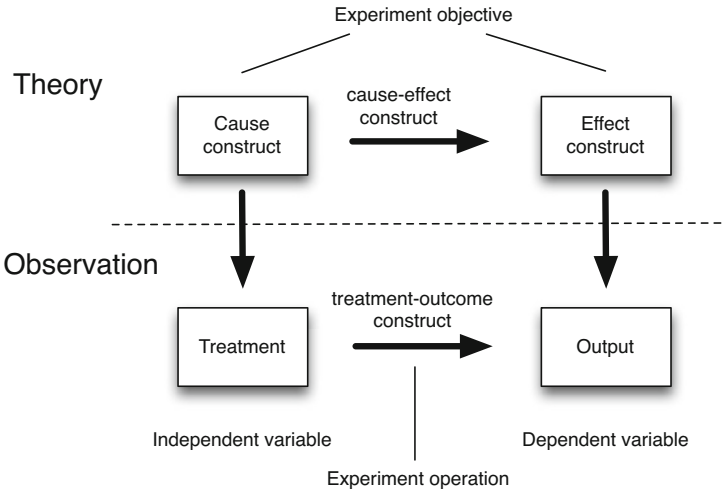


Fig. 6.1 Experiment principles. (Adapted from Trochim [253])

In order to evaluate our beliefs, we may use an experiment. The experiment is created, for example, to test a theory or hypothesis. In the design of the experiment, we have a number of treatments (values that the studied variable can take; see below) over which we have control. The experiment is performed and we are able to observe the outcome. This means that we test the relationship between the treatment and the outcome. If the experiment is properly set up, we should be able to draw conclusions about the relationship between the cause and the effect for which we stated a hypothesis.

The main objective of an experiment is to evaluate a hypothesis or relationship (see also the aspects listed above). The model may be derived using multivariate statistical methods, for example, regression techniques, and then we evaluate it in an experiment. Multivariate statistical methods are treated by, for example, Kachigan [124, 125] and Manly [165]. The focus in this book is primarily on hypothesis testing. Methods for statistical inference are applied with the purpose of showing with statistical significance that one method is better than the other [180, 207, 228]. The statistical methods are further discussed in Chap. 11.

6.2 Variables, Treatments, Objects, and Subjects

Before discussing the experiment process in some more detail, we introduce a few definitions to have a vocabulary for experimentation. When conducting a controlled experiment, we want to study the outcome when we vary some of the input variables to a process. There are two kinds of variables in an experiment, independent and dependent variables (see Fig. 6.2).

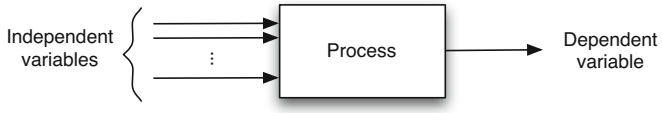


Fig. 6.2 Illustration of independent and dependent variables

Those variables that we want to study to see the effect of the changes in the independent variables are called *dependent variables* (or response variables). Often there is only one dependent variable in an experiment. All variables in a process that are manipulated and controlled are called *independent variables*.

Example We want to study the effect of a new development method on the productivity of the personnel. We may have chosen to introduce an object-oriented design method instead of a function-oriented approach. The *dependent variable* in the experiment is the productivity. *Independent variables* may be the development method, the experience of the personnel, tool support, and the environment.

An experiment studies the effect of changing one or more independent variables. Those variables are called *factors*. The other independent variables are controlled at a fixed level during the experiment, or else we cannot say if the factor or another variable causes the effect. A *treatment* is one particular value of a factor.

Example The factor for the example experiment above is the development method since we want to study the effect of changing the method. We use two treatments of the factor: the old and the new development method.

The choice of treatment, and at which levels the other independent variables shall be set, is part of the experiment design (see Fig. 6.3). Experiment design is described in more detail in Chap. 9.

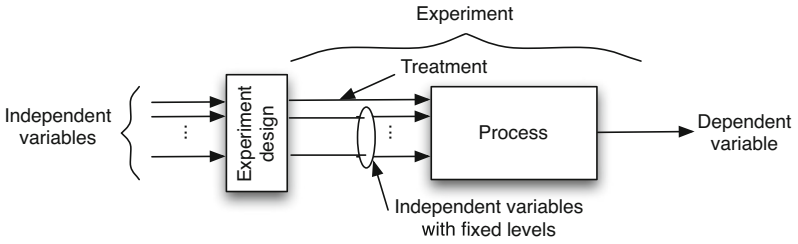


Fig. 6.3 Illustration of an experiment

The treatments are being applied to the combination of *objects* and *subjects*. An object can, for example, be a document that shall be reviewed with different inspection techniques. The people that apply the treatment are called *subjects*.[1] The characteristics of both the objects and the subjects can be independent variables in the experiment.

Example The *objects* in the example experiment are the programs to be developed and the *subjects* are the personnel.

An experiment consists of a set of *tests* (sometimes called trials) where each test is a combination of treatment, subject, and object. It should be observed that this type of test should not be mixed up with the use of statistical tests, which is further discussed in Chap. 11. The number of tests affects the experimental error, and provides an opportunity to estimate the mean effect of any experimental factor. The experimental error helps us to know how much confidence we can place in the results of the experiment.

Example A *test* can be that person *N* (*subject*) uses the new development method (*treatment*) for developing program *A* (*object*).

In human-oriented experiments, humans are the subjects, applying different treatments to objects. This implies several limitations to the control of the experiment. First, humans have different skills and abilities, which in itself may be an independent variable. Second, humans learn over time, which means that if one subject applies two methods, the order of application of the methods may matter, and also the same object cannot be used for both occasions. Third, human-oriented experiments are impacted by all sorts of influences and threats, due to the subject's ability to guess what the experimenter expects, their motivation for doing the tasks, etc. Hence, how subjects are selected and treated is critical for the outcome of the experiment.

Technology-oriented experiments are easier to control, since the technology may be made deterministic. The independent variable out of control in this type of experiments may instead be the objects selected for the experiment. One tool or technique may be well suited for one type of program, and not for another. Hence it is critical for the outcome how objects are selected.

[1] Sometimes the term *participant* is used instead of the term subject. The term subject is mainly used when people are considered with respect to different treatments and with respect to the analysis and the term participant mainly when it deals with how to engage and motivate people in a study.

6.3 Process

A process provides steps that support an activity, for example, software development. Processes are important as they can be used as checklists and guidelines for what to do and how to do it. To perform an experiment, several steps have to be taken and they have to be in a certain order. Thus, a process for how to perform experiments is needed.

The process presented is focused on experimentation, but the same basic steps must be performed in any empirical study. The main difference is the work within a specific activity. For example, the design of a survey, experiment, and case study differs, but they all need to be designed. Furthermore, as case studies are flexible design studies, there are several iterations over the process steps, while experiments and surveys, as fixed design studies, primarily execute the steps once. Thus, the basic process may be used for other types of studies than experiments, but it has to be tailored to the specific type of study being conducted, for example, a survey using email or a case study of a large software project. However, the process as it is presented is suited for both randomized experiments and quasi-experiments. The latter are often used in software engineering when, for example, random samples of subjects (participants) are infeasible.

The starting point for an experiment is that it has been identified as suitable using the decision structure presented in Sect. 2. In other words, we have to realize that an experiment is appropriate for the question we are going to investigate. This is by no means always obvious. However, empirical studies in software engineering have become more common. Different conclusions are reached by studies published before the millennium [252, 284] compared to a study published in 2018 [286]. If we assume that we have realized that an experiment is appropriate, then it is important to plan the experiment carefully to avoid unnecessary mistakes (see Sect. 3.5).

The experiment process can be divided into the following main steps. *Scoping* is the first step, where we scope the experiment in terms of problem, objective, and goals. *Planning* comes next, where the design of the experiment is determined, the instrumentation is considered, and the threats to the experiment are evaluated. *Operation* of the experiment follows from the design. In the operation step, measurements are collected which are analyzed and evaluated in *analysis and interpretation*. Finally, the results are *presented and packaged*. The steps are illustrated in Fig. 6.4 and further elaborated below, and then each of the steps is treated in-depth in Chaps. 8–12. An overview of the experiment process, including the steps and their sub-steps, is presented in Fig. 6.5.

The process is not supposed to be a "true" waterfall model; it is not assumed that a step is necessarily finished prior to the start of the next step. The order of steps in the process primarily indicates the starting order of the steps. In other words, the process is partly iterative and it may be necessary to go back and refine a previous step before continuing with the experiment. The main exception is that when the operation of the experiment has started, it is not possible to go back to the scoping and planning of the experiment. This is not possible since starting the operation

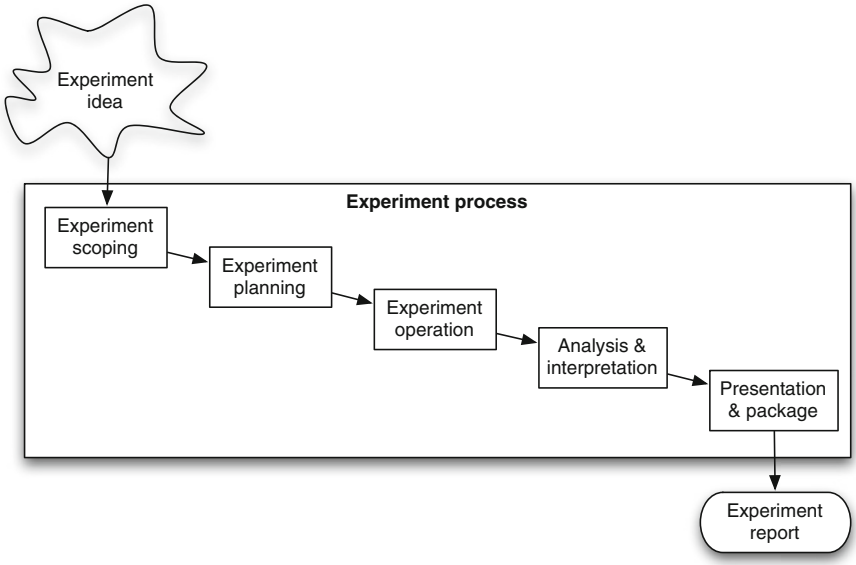


Fig. 6.4 Overview of the experiment process

means that the subjects are influenced by the experiment, and if we go back there is a risk that it will be impossible to use the same subjects when returning to the operation step of the experiment process. The five steps in the experimental process are described briefly as follows, and further elaborated in the chapters in Part II.

Scoping The first step is scoping. The hypothesis has to be stated clearly. It does not have to be stated formally at this stage, but it has to be clear. Furthermore, the objective and goals of the experiment must be defined. The goal is formulated from the problem to be solved. In order to capture the scope, a framework has been suggested [21]. The framework consists of the following constituents, which are further discussed in Chap. 8:

- Object of study (What is studied?)
- Purpose (What is the intention?)
- Quality focus (Which effect is studied?)
- Perspective (Whose view?)
- Context (Where is the study conducted?)

Planning The planning step is where the foundation of the experiment is laid. The *context* of the experiment is determined in detail. This includes personnel and the environment, for example, whether the experiment is run in a university environment with students or in an industrial setting. Moreover, the *hypothesis* of the experiment is stated formally, including a null hypothesis and an alternative hypothesis.

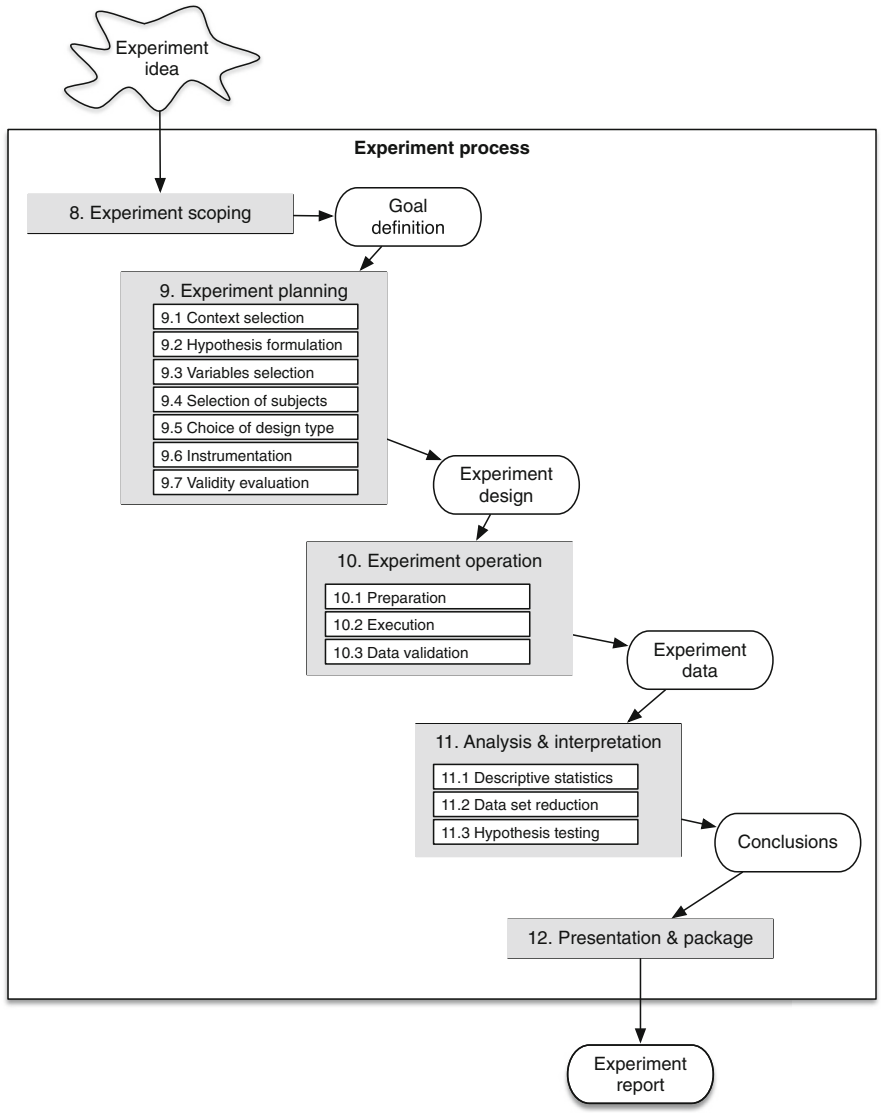


Fig. 6.5 Overview of the experiment process and artifacts with references to chapters and sections of this book

The next sub-step in the planning step is to determine *variables* (both independent variables (inputs) and the dependent variables (outputs)). An important issue regarding the variables is to determine the values the variables actually can take. This also includes determining the measurement scale, which puts constraints on the method that we later can apply for statistical analysis. The *subjects* of the study are identified.

Furthermore, the experiment is designed, which includes choosing a suitable *experiment design* including, for example, randomization of subjects. An issue closely related to the design is to prepare for the *instrumentation* of the experiment. We must identify and prepare suitable objects, develop guidelines if necessary, and define measurement procedures. These issues are further discussed in Chap. 9.

As a part of the planning, it is important to consider the question of *validity* of the expected results and design the experiment accordingly. Validity can be divided into four major classes: internal, external, construct, and conclusion validity. Internal validity is concerned with the validity within the given environment and the reliability of the results. External validity is a question of how general the findings are. Often, we would like to state that the results from an experiment are valid outside the actual context in which the experiment was run. Construct validity is a matter of judging if the treatment reflects the cause construct and the outcome provides a true picture of the effect construct (see Fig. 6.1). Conclusion validity is concerned with the relationship between the treatment and the outcome of the experiment. We have to judge if there is a relationship between the treatment and the outcome.

The planning is a crucial step in an experiment to ensure that the results from the experiment become useful. Poor planning may ruin a well-intended study.

Operation In principle, the operation consists of three sub-steps: preparation, execution, and data validation. In the *preparation* sub-step, we are concerned with preparing the subjects as well as the material needed, for example, data collection forms. The participants must be informed about the intention; we must have their consent and they must be committed. The actual *execution* is normally not a major problem. The main concern is to ensure that the experiment is conducted according to the plan and design of the experiment, which includes data collection. Finally, we must try to make sure that the actually collected *data* is correct and provide a *valid* picture of the experiment. The operation step is discussed in Chap. 10.

Analysis and Interpretation The data collected during operation provide the input to this step. The data can now be analyzed and interpreted. The first sub-step in the analysis and interpretation step is to try to understand the data by using *descriptive statistics*. These provide a visualization of the data. The descriptive statistics help us to understand and interpret the data informally.

The next sub-step is to consider whether the *data set* should be *reduced*, either by removing data points or by reducing the number of variables by studying if some of the variables provide the same information. Specific methods are available for data reduction.

After having removed data points or reduced the data set, we are able to perform a *hypothesis test*, where the actual test is chosen based on measurement scales, values on the input data, and the type of results we are looking for. The statistical tests together with a more detailed discussion of descriptive statistics and data reduction techniques can be found in Chap. 11.

One important aspect of this step is the interpretation. That is, we have to determine from the analysis whether the null hypothesis could be rejected in favor of the alternative hypothesis. This forms the basis for decision-making and conclusions concerning how to use the results from the experiment, which includes motivation for further studies, for example, to conduct an enlarged experiment or a case study.

Presentation and Package The final step is concerned with presenting and packaging the findings. This includes primarily documentation of the results, which can be made through a research paper for publication, a lab package for replication purposes, or as part of a company's experience base. This final step is essential to make sure that the lessons learned are taken care of in an appropriate way. Moreover, an experiment will never provide the final answer to a question, and hence it is important to facilitate replication of the experiment. A comprehensive and thorough documentation is a prerequisite to achieve this objective. Having said that, lab packages should be used with care since using the same experimental design and documents may carry over some systematic problems and biases from the original experiment, as discussed in Sect. 3.2. Independently, we must take some time after the experiment to document and present it in a proper way. The presentation of an experiment is further elaborated in Chap. 12.

6.4 Overview

The steps in the experiment process are described in more detail subsequently (Chaps. 8–12), and illustrated in Fig. 6.5. Moreover, to support the understanding of the process, an example of an experiment is presented in Chap. 13. The objective of the example is to closely follow the defined process to illustrate its use. Moreover, a published experiment is presented Chap. 14.

6.5 Exercises

6.1 What is a cause and effect relationship?

6.2 What is a treatment, and why is it sometimes necessary to apply treatments in a random order?

6.3 What are dependent and independent variables respectively?

6.4 What are quasi-experiments and A/B testing? Explain why these are essential and common in software engineering.

6.5 Which are the main steps in the experiment process, and why is it important to have distinct steps?

Chapter 7
Case Studies

The term "case study" appears every now and then in the title or in the abstracts of software engineering research papers. However, the presented studies range from very ambitious and well-organized studies in the field, to small toy examples that claim to be case studies. The latter should preferably be termed small-scale evaluation, examples, or illustrations. As mentioned in Sect. 2.8, the problem with the usage of the term "case study" is discussed in the literature.

Moreover, there are different taxonomies used to classify research. The term case study is used in parallel with terms like field study and observational study, each focusing on a particular aspect of the research methodology. For example, Lethbridge et al. use *field studies* as the most general term [157], while Easterbrook et al. call *case studies* one of five "classes of research methods" [65]. Zelkowitz and Wallace propose a terminology that is somewhat different from what is used in other fields, and categorize project monitoring, case study, and field study as *observational methods* [284]. This plethora of terms causes confusion and problems when trying to aggregate multiple empirical studies.

We have chosen to view "case study" as a research methodology since it is not a data collection method in itself. Case study research includes research methods for data collection. Furthermore, case study research is a methodology where key factors that may have any effect on the outcome are identified and then documented [240, 281]. Case study research is an observational methodology, i.e., it is done by observation of a real-life entity of analysis, for example, a development project, activity, task, or assignment.

Three commonly used definitions of case study research are provided by Robson [207], Yin [281], and Benbasat et al. [32]. The three definitions agree that case study is an empirical research approach aimed at *investigating contemporary phenomena* in their context. Robson calls it a research strategy and stresses the use of *multiple sources of evidence*, Yin denotes it an inquiry and remarks that *the boundary between the phenomenon and its context may be unclear*, while Benbasat et al. make the definitions somewhat more specific, mentioning *infor-*

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_7

mation gathering from a few entities (people, groups, organizations), and the *lack of experimental control*. Runeson et al. [212] provide a definition from a software engineering perspective:

Definition 7.1 **Case study** is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified [212].

The definition by Runeson et al. [212] is based on the definitions by Yin [281], Benbasat et al. [32], and Robson [207]. Wohlin [269] suggests a revised definition of *case study* based on the definition above by Runeson et al. [212]. The revision stresses the need for more than a single data collection *method* for case studies. Multiple *sources* of evidence may be misinterpreted, for example, as having interviewed several people. The objective of the revised definition is to clarify that *more than one data collection method* is expected according to the definition of case study. In addition, Wohlin also added a formulation to separate case studies from action research. The ACM Empirical Standard [204] and Wohlin and Runeson [272] treat case study research and action research as two separate research methodologies, which aligns with the definition by Wohlin [269]:

Definition 7.2 A **case study** is an empirical investigation of a case, using multiple data collection methods, to study a contemporary phenomenon in its real-life context, and with the investigator(s) not taking an active role in the case investigated [269].

The definition by Wohlin [269] includes the following five essential components of a case study:

- Empirical investigation of a case
- Multiple data collection methods
- Contemporary phenomenon
- Real-life context
- Investigator's role

As highlighted in the definition by Wohlin [269], action research is closely related to case study research with its purpose to "influence or change some aspect of whatever is the focus of the research" [207]. More strictly, a case study is purely observational while action research is focused on and involved in the change process. In software process improvement [59, 117] and technology transfer studies [92], the research approach could be characterized as action research if the researcher actively participates in the improvements. However, when studying the effects of a change, for example, in pre- and post-event studies, we classify the approach as case study. In information system research, where action research is widely used, there is a discussion on finding the balance between action and research (see, e.g., Baskerville and Wood-Harper [31] or Avison et al. [11]). For the research part of action research, these guidelines for case studies may be used too.

Staron [242] discusses further details of action research in a software engineering context.

Easterbrook et al. [65] also count ethnographic studies among the major research methodologies. We prefer to consider ethnographic studies as a specialized type of case studies with focus on cultural practices [65] or long duration studies with large amounts of participant-observer data [137]. This is aligned with the ACM Empirical Standards [204], although not with the viewpoint of Sharp et al. [219]. Zelkowitz and Wallace define four different "observational methods" in software engineering [284]: project monitoring, case study, assertion, and field study. We prefer to see project monitoring as a part of a case study and field studies as multiple case studies, while assertion is not considered an accepted research method.

Robson summarizes his view, which seems functional in software engineering as well: "Many flexible design studies, although not explicitly labeled as such, can be usefully viewed as case studies" [207].

A case study should include more than one research method; for example, a survey may be conducted within a case study, and archival analyses may be a part of its data collection. Ethnographic methods, like interviews and observations, are also frequently used for data collection in case studies.

Yin adds specifically to the characteristics of a case study that it [281]:

- "Copes with the technically distinctive situation in which there will be many more variables than data points, and as one result,
- Relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result,
- Benefits from the prior development of theoretical propositions to guide data collection and analysis."

Hence, a case study will never provide conclusions with statistical significance. On the contrary, many different kinds of evidence, figures, statements, and artifacts are linked together to support strong and relevant conclusions.

In summary, the key characteristics of a case study are that [212]:

1. It is of flexible type, coping with the complex and dynamic characteristics of real-world phenomena, like software engineering.
2. Its conclusions are based on a clear chain of evidence, whether qualitative or quantitative, collected from multiple sources in a planned and consistent manner.
3. It adds to existing knowledge by being based on previously established theory, if such exists, or by building theory.

7.1 Why Case Studies in Software Engineering?

The area of software engineering involves development, operation, and maintenance of software and related artifacts. Research on software engineering is to a large extent aimed at investigating how development, operation, and maintenance are

conducted by software engineers and other stakeholders under different conditions. Individuals, groups, and organizations carry out software development, and social and political questions are of importance for this development. Software development is a matter of balancing human, social, and organizational capitals [276]. That is, software engineering is a multidisciplinary field involving areas where case studies are conducted, like psychology, sociology, political science, social work, business, and community planning (e.g., [170, 281]). This means that many research questions in software engineering are suitable for case study research.

The definition of case study at the beginning of this chapter focuses on studying phenomena in their context, particularly when the boundary between the phenomenon and its context is unclear. This is true in software engineering. Experimentation in software engineering has clearly shown that there are many factors impacting the outcome of a software engineering activity, for example, when trying to replicate studies (see Sect. 3.2). Case studies offer an approach that does not need a strict boundary between the studied object and its environment. Perhaps the key to understanding is in the interaction between the two?

The purpose of the study may, for example, be explanatory, descriptive, exploratory, or evaluation (see Fig. 2.1). Case studies are very suitable for industrial evaluation of software engineering methods and tools because they can avoid scale-up problems that may come with studies conducted in a laboratory setting. The use of case studies in industry–academia collaboration is further discussed by Wohlin and Runeson [272]. If the effect of, for example, a method or tool change is very widespread within a company, a case study may be more suitable than, for example, an experiment. The effect of the change can only be assessed at a high level of abstraction because the process change includes smaller and more detailed changes throughout the development process [136]. Also, the effects of the change cannot be identified immediately. For example, if we would like to know if a new design tool increases the reliability, it may be necessary to wait until after delivery of the developed product to assess the effects on operational failures.

Case studies may be used to study, for example, development projects, activities, tasks, or assignments. Data is collected for a specific purpose throughout the study and may be quantitative or qualitative. Based on the data collection, statistical analyses can be carried out for the quantitative or quantified data, i.e., typically qualitative data that has been assigned to different types or categories. The case study is normally aimed at tracking a specific attribute or establishing relationships between different attributes. The level of control is lower in a case study than in an experiment. A case study may, for example, be aimed at building a model to predict the number of faults in testing [6]. Multivariate statistical analysis is often applied in this type of study. The analysis methods include linear regression and principal component analysis [165]. Furthermore, case studies may include qualitative analysis. For example, Petersen and Wohlin [191] performed interviews to identify concerns and improvements in the software development as a company moved from a plan-driven approach to being more agile and incremental in their development. The analysis was done using content analysis of the transcribed interviews. Case study research is further discussed in general by, for example,

Robson [207], Stake [240], and Yin [281], and specifically for software engineering by Pfleeger [194], Kitchenham et al. [136], Verner et al. [260], Runeson and Höst [211], and Runeson et al. [212].

If, for example, we would like to compare two methods, the study may be defined as a case study or an experiment, depending on the scale of the evaluation, the ability to isolate factors, feasibility for randomization, and the setting of the study. Case studies are conducted in a real-life context, while experiments may be conducted in an industrial setting. The difference between case studies and experiments is that experiments sample over the variables that are being manipulated, while case studies select from the variables representing the typical situation. Thus, there is a level of control in experiments that we do not have for case studies. An advantage of case studies is that they are easier to plan and are more realistic, but can be harder to conduct given the dependence on the setting in which it is performed. Furthermore, a disadvantage is that the results are difficult to generalize and harder to interpret, i.e., it is possible to show the effects in a typical situation, but it requires more analysis to generalize to other situations [281].

Case study research is a standard method used for empirical studies in various sciences such as sociology, medicine, and psychology. However, as they are different from analytical and controlled empirical studies, case studies have been criticized for being of less value, being impossible to generalize from, being biased by researchers, etc. The critique may be addressed by applying proper research methodology practices and accepting that knowledge is not only about statistical significance [80, 155].

7.2 Case Study Arrangements

A case study can be applied as a comparative research approach, comparing the results of using one method or some form of manipulation to the results of using another approach. To avoid bias and to ensure internal validity, it is necessary to create a solid base for assessing the results of the case study. Kitchenham et al. propose three ways to arrange the study to facilitate this [136]:

- A comparison of the results of using the new method against a company baseline is one solution. The company should gather data from standard projects and calculate characteristics like average productivity and defect rate. Then it is possible to compare the results from the case study with the figures from the baseline.
- A sister project can be chosen as a baseline. The project under study uses the new method and the sister project the current one. Both projects should have the same characteristics, i.e., the projects must be comparable.
- If the method applies to individual product components, it could be applied at random to some components and not to others. This is very similar to an experiment, but since the projects are not drawn at random from the population of all projects, it is not an experiment.

If the above is infeasible, we may have to settle for a more qualitative comparison. For example, assume that we want to conduct a case study concerning the use of a new test method. The conjecture is that the new test method increases the quality of the resulting product, for example, fewer failures in operation. It may be hard to verify that the new test method reduces the number of failures in operation, since this requires a reference system that does not use the new test method. To verify that one test method is better than the other, there is a need to use two different test methods on the same system, or systems will have very similar characteristics. This is in most cases infeasible due to economic constraints. Moreover, it is insufficient to compare with the outcome for an older system using a different test method. The older system is different, other developers and testers may have worked with the old system, and many other characteristics may be different. Thus, we may have to interview developers and testers to get their perception of the performance of the two test methods.

7.3 Confounding Factors and Other Aspects

When performing case studies it is necessary to minimize the effects of confounding factors. A confounding factor is a factor that makes it impossible to distinguish the effects of two factors from each other. This is important since we do not have the same control over a case study as in an experiment. For example, it may be difficult to tell if a better result depends on the tool or the experience of the user of the tool. Confounding effects could involve problems with learning how to use a tool or method when trying to assess its benefits, or using very enthusiastic or skeptical staff.

There are both pros and cons with case studies. Case studies are valuable because they incorporate qualities that an experiment cannot visualize, for example, scale, complexity, unpredictability, and dynamics. Some potential issues with case studies are as follows:

- A small or simplified case study is seldom a good instrument for discovering software engineering principles and techniques. Increases in scale lead to changes in the types of problems that become most indicative. In other words, the problem may be different in a small case study and in a large case study, although the objective is to study the same issues. For example, in a small case study the main problem may be the actual technique being studied, and in a large case study the major problem may be the number of people involved and hence also the communication between people.
- Researchers are not in full control of a case study situation. This is good, from one perspective, because unpredictable changes frequently tell them much about the problems being studied. The problem is that we cannot be sure about the effects due to confounding factors.

7.4 Case Study Research Process

When conducting a case study, there are four major process steps to go through:

1. Case study design and planning: objectives are defined and the case study is planned.
2. Preparation and data collection: procedures and protocols for data collection are defined, after which the data is collected from the case studies.
3. Data and validity analysis.
4. Reporting.

This process is almost the same for any kind of empirical study; compare this, for example, to the process outlined in Chap. 6 and further elaborated in Chaps. 8–12 for experiments and Kitchenham et al. [137]. However, as case study methodology is a flexible design, there is a significant amount of iteration over the steps [6]. The data collection and analysis may be conducted incrementally. If insufficient data is collected for the analysis, more data collection may be planned, etc. Eisenhardt adds two steps between 3 and 4 above in her process for building theories from case study research [67], (a) shaping hypotheses and (b) enfolding literature, while the remainder is basically the same, except that terminological variations are the same as above. The process steps are presented in Sects. 7.5–7.8.

7.5 Design and Planning

Case study research is of flexible type but this does not mean that planning is unnecessary. On the contrary, good planning for a case study is crucial for its success. There are several issues that need to be planned, such as what methods to use for data collection, what departments of an organization to visit, what documents to read, which persons to interview, how often interviews should be conducted, etc. These plans can be formulated in a case study protocol (see Sect. 7.5.2).

7.5.1 Case Study Planning

A plan for a case study should at least contain the following elements [207]:

- *Objective:* What do we want to achieve?
- *The case:* What is being studied?
- *Theory:* What is the frame of reference?
- *Research questions:* What do we want to learn?
- *Methods:* How do we collect data?
- *Selection strategy:* Where do we seek data?

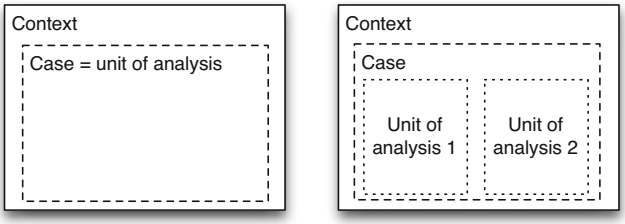


Fig. 7.1 Holistic case study (left) and embedded case study (right)

The objective is naturally more generally formulated and less precise than in fixed research designs. The objective is initially more like a focus point that evolves during the study. The research questions state what needs to be known to fulfill the objective of the study. Similar to the objective, the research questions evolve during the study and are narrowed down to specific research questions during the study iterations [6].

In software engineering, the case may be a software development project, which is the most straightforward choice. It may alternatively be an individual, a group of people, a process, a product, a policy, a role in the organization, an event, a technology, etc. The project, individual, group, etc. may also constitute a unit of analysis within a case. Studies of "toy programs" or similar are of course excluded due to their lack of real-life context.

Yin [281] distinguishes between holistic case studies, where the case is studied as a whole, and embedded case studies, where multiple units of analysis are studied within a case (see Fig. 7.1). Whether to define a study consisting of two cases as holistic or embedded depends on what we define as the context and research goals. For example, if studying two projects in two different companies and in two different application domains, both using agile practices, it could be discussed if it is one or two studies. On the one hand, the projects may be considered two units of analysis in an embedded case study if the context is software companies in general and the research goal is to study agile practices. On the other hand, if the context is considered the specific company or application domain, they have to be seen as two separate holistic cases.

Using theories to develop the research direction is not well established in the software engineering field, as discussed in Sect. 3.3. However, defining the frame of reference of the study makes the context of the case study research clear, and helps both those conducting the research and those reviewing its results. In lack of theory, the frame of reference may alternatively be expressed in terms of the viewpoint taken in the research and the background of the researchers. Case studies based on grounded theory naturally have no specified theory [51].

The principal decisions on methods for data collection are defined at design time for the case study, although detailed decisions on data collection procedures are taken later. Lethbridge et al. [157] define three categories of methods: direct (e.g.,

interviews), indirect (e.g., tool instrumentation), and independent (e.g., documentation analysis). These are further elaborated in Sect. 7.6.

In case studies, the case and the units of analysis should be selected intentionally. This is in contrast to surveys and experiments, where subjects are sampled from a population to which the results are intended to be generalized. The purpose of the selection may be to study a case that is expected to be "typical," "critical," "revelatory," or "unique" in some respect [32], and the case is selected accordingly. In a comparative case study, the units of analysis must be selected to have the variation in properties that the study intends to compare. However, in practice, many cases are selected based on availability [32], which is similar for experiments [234].

Case selection is particularly important when replicating case studies. A case study may be literally replicated, i.e., the case is selected to predict similar results, or it is theoretically replicated, i.e., the case is selected to predict contrasting results for predictable reasons [281].

7.5.2 Case Study Protocol

The case study protocol is a container for the design decisions on the case study as well as field procedures for carrying through the study. The protocol is a continuously changed document that is updated when the plans for the case study are changed and serves several purposes:

1. It serves as a guide when conducting the data collection, and in that way prevents the researcher from missing out on collecting data that were planned to be collected.
2. The processes of formulating the protocol make the research concrete in the planning step, which may help the researcher to decide what data sources to use and what questions to ask.
3. Other researchers and relevant stakeholders may review it to give feedback on the plans. Feedback on the protocol from other researchers can, for example, lower the risk of missing relevant data sources, interview questions, or roles to include in the research and can help to question the relation between research questions and interview questions.
4. It can serve as a log or diary where all data collection and analysis is recorded together with change decisions based on the flexible nature of the research. This can be an important source of information when the case study is reported later on. In order to keep track of changes during the research project, the protocol should be kept under some form of version control.

Brereton et al. [37] propose an outline of a case study protocol, which is summarized in Table 7.1. As the proposal shows, the protocol is quite detailed to support a well-structured research approach.

Table 7.1 Outline of case study protocol according to Brereton et al. [37]

Section	Content
Background	Previous research, main and additional research questions
Design	Single or multiple cases, embedded or holistic design; object of study; propositions derived from research questions
Selection	Criteria for case selection
Procedures and roles	Field procedures; roles for research team members
Data collection	Identify data, define collection plan and data storage
Analysis	Criteria for interpretation, linking between data and research questions, alternative explanations
Plan validity	Tactics to reduce threats to validity
Study limitations	Specify remaining validity issues
Reporting	Target audience
Schedule	Estimates for the major steps
Appendices	Any detailed information

7.6 Preparation and Data Collection

There are several different sources of information that can be used in a case study. It is important to use several data sources in a case study to limit the effects of one interpretation of one single data source. Moreover, it does not qualify as a case study if one data source implies using only one data collection method. If the same conclusion can be drawn from several sources of information, i.e., triangulation, the conclusions are stronger than conclusions based on a single source. In a case study, it is also important to take into account viewpoints of different roles, and to investigate differences, for example, between different projects and products. Commonly, conclusions are drawn by analyzing differences between data sources.

According to Lethbridge et al. [157], data collection techniques can be divided into three levels:

- *First degree:* Direct methods means that the researcher is in direct contact with the subjects and collects data in real time. This is the case with, for example, interviews, focus groups, Delphi surveys [55], and observations with "think aloud protocols" [188].
- *Second degree:* Indirect methods where the researcher directly collects raw data without actually interacting with the subjects during the data collection. Examples are logging of the usage of software engineering tools, and observations through video recording.
- *Third degree:* Independent analysis of work artifacts where already available and sometimes compiled data is used. This is, for example, the case when documents such as requirements specifications and failure reports from an organization are analyzed or when data from organizational databases such as time accounting is analyzed.

First degree methods are generally more expensive to apply than second or third degree methods, since they require significant effort both from the researcher and the subjects. An advantage of the first and second degree methods is that the researcher can to a large extent precisely control what data is collected, how it is collected, in what form the data is collected, the context, etc. Third degree methods are generally less expensive, but they do not offer the same control to the researcher; hence the quality of the data is not under control either, neither regarding the original data quality nor its use for the case study purpose. In many cases the researcher must, to some extent, base the details of the data collection on what data is available. For third degree methods, it should also be noted that the data has been collected and recorded for another purpose than that of the research study, contrary to general metrics guidelines [257]. It is not certain that requirements on data validity and completeness were the same when the data was collected, as they are in the research study.

In Sects. 7.6.1–7.6.4, we discuss three specific data collection methods and metrics. The focus is on methods that we have found useful in software engineering case studies, i.e., interviews, observations, and archival analysis, as well as metrics [32, 212, 281].

7.6.1 Interviews

In interview-based data collection, the researcher asks a series of questions to a set of subjects about the areas of interest in the case study. In most cases one interview is conducted with every single subject, but it is possible to conduct group interviews, for example, focus groups [149]. The dialogue between the researcher and the subject(s) is guided by a set of interview questions.

The interview questions are based on the research questions (although not phrased in the same way). Questions can be *open*, i.e., allowing and inviting a broad range of answers and issues from the interviewed subject, or *closed*, i.e., offering a limited set of alternative answers.

Interviews can be divided into *unstructured, semi-structured* and *fully structured* interviews [207]. In an unstructured interview, the interview questions are formulated as general concerns and interests from the researcher. In this case the interview conversation will develop based on the interest of the subject and the researcher. In a fully structured interview all questions are planned in advance and all questions are asked in the same order as in the plan. In many ways, a fully structured interview is similar to a questionnaire-based survey. In a semi-structured interview, questions are planned, but they are not necessarily asked in the same order as they are listed. The development of the conversation in the interview can decide in which order the different questions are handled, and the researcher can use the list of questions to be certain that all questions are handled, i.e., more or less as a checklist. Additionally, semi-structured interviews allow for improvisation and exploration of the studied

Table 7.2 Overview of interview types

	Unstructured	**Semi-structured**	**Fully structured**
Typical foci	How individuals qualitatively experience the phenomenon	How individuals qualitatively and quantitatively experience the phenomenon	Researcher seeks to find relations between constructs
Interview questions	Interview guide with areas to focus on	Mix of open and closed questions	Closed questions
Objective	Exploratory	Descriptive and explanatory	Descriptive and explanatory

objects. Semi-structured interviews are common in case studies. The three types of interviews are summarized in Table 7.2.

An interview session may be divided into a number of activities. First the researcher presents the objectives of the interview and the case study, and explains how the data from the interview will be used. Then a set of introductory questions is asked about the background, etc. of the subject; these questions are relatively simple to answer. After the introduction, the main interview questions are posed, which take up the largest part of the interview. If the interview contains personal and perhaps sensitive questions, for example, concerning economy, opinions about colleagues, why things went wrong, or questions related to the interviewee's own competence [111], it is important that the interviewee is ensured confidentiality and that the interviewee trusts the interviewer. It is not recommended to start the interview with these questions or to introduce them before a climate of trust has been obtained. It is recommended that the researcher summarizes the major findings toward the end of the interview to get feedback and avoid misunderstandings.

During the interview sessions, it is recommended to record the discussion in a suitable audio or video format. Even if notes are taken, it is in many cases hard to record all details, and it is impossible to know what is important to record during the interview. When the interview has been recorded it needs to be transcribed into text before it is analyzed. In some cases it may be advantageous to have the transcripts reviewed by the interview subject. Strandberg provides practical guidelines for interview data collection and analysis, with a particular focus on ethical aspects [248].

During the planning step of an interview study it is decided whom to interview. Due to the qualitative nature of an interview study it is recommended to select subjects based on differences instead of trying to replicate similarities, as discussed in Sect. 7.5. This means that it is good to try to involve different roles, personalities, etc. in the interview. The number of interviewees has to be decided during the study. One criterion for when sufficient interviews are conducted is "saturation", i.e., when no new information or viewpoint is gained from new subjects [51].

7.6.2 Observations

Observations can be conducted to investigate how software engineers conduct a certain task. There are many different approaches for observation. One approach is to monitor a group of software engineers with a video recorder and later on analyze the recording. Another alternative is to apply a "think aloud" protocol, where the researcher is repeatedly asking questions like "What is your strategy?" and "What are you thinking?" to remind the subjects to think aloud. This can be combined with recording of audio and keystrokes as proposed, for example, by Wallace et al. [265]. Observations in meetings are another type, where meeting attendants interact with each other, and thus generate information about the studied object. Karahasanovic̈ et al. [129] present an alternative approach where a tool for sampling is used to obtain data and feedback from the participants.

Approaches for observations can be divided into high or low interaction of the researcher and high or low awareness of the subjects being observed (see Table 7.3).

Observations according to category 1 or category 2 are typically conducted in action research or classical ethnographic studies where the researcher is part of the team, and not only seen as a researcher by the other team members. The difference between category 1 and category 2 is that in category 1 the researcher is seen as an "observing participant" by the other subjects, while she is more seen as a "normal participant" in category 2. In category 3, the researcher is seen only as a researcher. The approaches for observation typically include observations with first degree data collection techniques, such as a "think aloud" protocol as described above. In category 4 the subjects are typically observed with a second degree technique such as video recording (sometimes called video ethnography).

An advantage of observations is that they may provide a deep understanding of the phenomenon that is studied. Furthermore, it is particularly relevant to use observations where it is suspected that there is a deviation between an "official" view of matters and the "real" case [205]. It should, however, be noted that it produces a substantial amount of data, which makes the analysis time-consuming.

Table 7.3 Different approaches to observations

	High awareness of being observed	Low awareness of being observed
High degree of interaction by the researcher	Category 1	Category 2
Low degree of interaction by the researcher	Category 3	Category 4

7.6.3 Archival Analysis

Data from archival analysis refers to, for example, meeting minutes, documents from different development phases, failure data, organizational charts, financial records, and other previously collected measurements in an organization.

The data from archival analysis is a third degree type of data that can be collected in a case study. For this type of data configuration, management tools are important sources, since they enable the collection of a number of different documents and different versions of documents. Mining Software Repositories (MSR) has evolved as a line of software engineering research in itself and has become popular thanks to easy access of open source software repositories [103, 261]. MSR (or archival analysis) could be one way of collecting data in a case study. However, many MSR studies, particularly studies of open source software, are incorrectly labeled as case studies [271]; it is insufficient to use archival analysis to label a study as a case study, due to reliance on a single source of information. It does not fulfill the definitions of case study research from several perspectives, including using one data collection method and the study not being conducted in a real-life context, unless the study also includes, for example, interviews or some other interaction with the developers. However, it should be noted that MSR research often includes case study characteristics, although many studies are erroneously labeled as experiments [12].

As for other third degree data sources, it is important to keep in mind that the artifacts were not originally developed with the intention to provide data for the research. It is of course hard for the researcher to assess the quality of the data, although some information can be obtained by investigating the purpose of the original data collection, and by interviewing relevant people in the organization.

7.6.4 Metrics

The abovementioned data collection techniques are mostly focused on qualitative data. However, quantitative data is also important in a case study. Collected data can either be defined or collected for the purpose of the case study, or already available data can be used in a case study. The first case gives, of course, the most flexibility and the data that is most suitable for the research questions under investigation. The definition of what data to collect should be based on a goal-oriented measurement technique, such as the Goal Question Metric method (GQM) [23, 257], which is presented in Sect. 3.5.

Examples of already available data are effort data from older projects, sales figures of products, metrics of product quality in terms of failures, etc. This kind of data may, for example, be available in a database in an organization. However, it should be noted that the researcher can neither control nor assess the quality of the

data, since it was collected for another purpose, and as for other forms of archival analysis, there is a risk of missing important data.

7.7 Data and Validity Analysis

7.7.1 Quantitative Data Analysis

Data analysis is conducted differently for quantitative and qualitative data. For quantitative data, the analysis typically includes analysis of descriptive statistics, correlation analysis, development of predictive models, and hypothesis testing. All of these activities are relevant in case study research. Quantitative data analysis, although primarily in an experimental context, is further described in Chap. 11.

Descriptive statistics, such as mean values, standard deviations, histograms, and scatter plots, are used to get an understanding of the data that has been collected. Correlation analysis and development of predictive models are conducted to describe how a measurement from a later process activity is related to an earlier process measurement. Hypothesis testing is conducted to determine if there is a significant effect of one or several variables (independent variables) on one or several other variables (dependent variables).

It should be noted that methods for quantitative analysis assume a fixed research design. For example, if a question with a quantitative answer is changed halfway in a series of interviews, this makes it impossible to interpret the mean value of the answers. Moreover, quantitative data sets from single cases tend to be very small, due to the number of respondents or measurement points, which causes particular concerns in the analysis.

7.7.2 Qualitative Data Analysis

The basic objective of the qualitative analysis is to derive conclusions from the data, keeping a clear chain of evidence. The chain of evidence means that a reader should be able to follow the derivation of results and conclusions from the collected data [281]. This means that sufficient information from each step of the study and every decision made by the researcher must be presented.

In addition, analysis of qualitative research is characterized by having analysis carried out in parallel with the data collection and the need for systematic analysis techniques. Analysis must be carried out in parallel with the data collection since the approach is flexible and new insights are found during the analysis. In order to investigate these insights, new data must often be collected, and instrumentation such as interview questionnaires must be updated. The need to be systematic is a

direct result of the fact that the data collection techniques can be constantly updated, while at the same time being required to maintain a chain of evidence.

In order to reduce bias by individual researchers, the analysis benefits from being conducted by multiple researchers. The preliminary results from each individual researcher are merged into a common analysis result in a second step. Keeping track and reporting the cooperation scheme helps in increasing the validity of the study.

General Techniques for Analysis There are two different parts of data analysis of qualitative data, hypothesis generating techniques and hypothesis confirmation techniques [217].

Hypothesis *generation* is intended to find hypotheses from the data. When using this kind of technique, the researcher should try to be unbiased and open to whatever hypotheses are to be found in the data. The results of these techniques are the hypotheses as such. Examples of hypotheses generating techniques are "constant comparisons" and "cross-case analysis" [217]. Hypothesis *confirmation* techniques denote techniques that can be used to confirm that a hypothesis is true, for example, through analysis of more data. Triangulation and replication are examples of approaches for hypothesis confirmation [217]. *Negative case analysis* tries to find alternative explanations that reject the hypotheses. These basic types of techniques are used iteratively and in combination. First hypotheses are generated and then they are confirmed. Hypothesis generation may take place within one cycle of a case study, or with data from one unit of analysis, and hypothesis confirmation may be done with data from another cycle or unit of analysis [6].

This means that analysis of qualitative data is conducted in a series of steps (based on Robson [207]). First the data is coded, which means that parts of the text can be given a code representing a certain theme, area, construct, etc. One code is usually assigned to many pieces of text, and one piece of text can be assigned more than one code. Codes can form a hierarchy of codes and sub-codes. The coded material can be combined with comments and reflections by the researcher (i.e., "memos"). When this has been done, the researcher can go through the material to identify a first set of hypotheses. This can, for example, be phrases that are similar in different parts of the material, patterns in the data, differences between sub-groups of subjects, etc. The identified hypotheses can then be used when further data collection is conducted in the field, i.e., resulting in an iterative approach where data collection and analysis is conducted in parallel as described above. During the iterative process a small set of generalizations can be formulated, eventually resulting in a formalized body of knowledge, which is the final result of the research attempt. This is, of course, not a simple sequence of steps. Instead, they are executed iteratively and they affect each other.

One example of a useful technique for analysis is tabulation, where the coded data is arranged in tables, which makes it possible to get an overview of the data. The data can, for example, be organized in a table where the rows represent codes of interest and the columns represent interview subjects. However, how to do this must be decided for every case study.

There are specialized software tools available to support qualitative data analysis, for example, NVivo[1] and Atlas.[2] However, in some cases standard tools such as word processors and spreadsheet tools are useful when managing the textual data.

Level of Formalism A structured approach is, as described above, important in qualitative analysis. However, the analysis can be conducted at different levels of formalism. Robson [207] mentions the following approaches:

- *Immersion approaches:* These are the least structured approaches, with a very low level of structure, more reliant on intuition and interpretive skills of the researcher. These approaches may be hard to combine with requirements on keeping and communicating a chain of evidence.
- *Editing approaches:* These approaches include a few a priori codes, i.e., codes are defined based on findings of the researcher during the analysis.
- *Template approaches:* These approaches are more formal and include a priori codes based on research questions.
- *Quasi-statistical approaches:* These approaches are much more formalized and include, for example, calculation of frequencies of words and phrases.

In our experience editing approaches and template approaches are most suitable in software engineering case studies. It is hard to present and obtain a clear chain of evidence in informal immersion approaches. It is also hard to interpret the result of, for example, frequencies of words in documents and interviews.

7.7.3 Validity

The validity of a study denotes the trustworthiness of the results, and to what extent the results are true and not biased by the researcher's subjective point of view. It is, of course, too late to consider the validity during the analysis. The validity must be addressed during all previous steps of the case study [259].

There are different ways to classify aspects of validity and threats to validity in the literature. Here we chose a classification scheme, which is also used by Yin [281] for case studies, and similar to what is usually used in controlled experiments in software engineering as further elaborated in Sect. 9.7. Some researchers have argued for having a different classification scheme for flexible design studies (credibility, transferability, dependability, and confirmability), while we prefer to operationalize this scheme for flexible design studies, instead of changing the terms [207]. This scheme distinguishes between four aspects of the validity, which

[1] https://lumivero.com/

[2] https://atlasti.com/

can be summarized as follows:

- *Construct validity:* This aspect of validity reflects to what extent the operational measures that are studied really represent what the researcher has in mind and what is investigated according to the research questions. If, for example, the constructs discussed in the interview questions are not interpreted in the same way by the researcher and the interviewed persons, there is a threat to construct validity [233].
- *Internal validity:* This aspect of validity is of concern when causal relations are examined. When the researcher is investigating whether one factor affects an investigated factor, there is a risk that the investigated factor is also affected by a third factor. If the researcher is not aware of the third factor and/or does not know to what extent it affects the investigated factor, there is a threat to internal validity.
- *External validity:* This aspect of validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case. During analysis of external validity, the researcher tries to analyze to what extent the findings are of relevance for other cases. In case studies, there is no population from which a statistically representative sample has been drawn. However, for case studies, the intention is to enable analytical generalization where the results are extended to cases which have common characteristics and hence for which the findings are relevant, i.e., defining a theory.
- *Reliability:* This aspect is concerned with the extent to which the data and the analysis are dependent on the specific researchers. Hypothetically, if another researcher later on conducted the same study, the result should be the same. Threats to this aspect of validity are, for example, if it is not clear how to code collected data or if questionnaires or interview questions are unclear. For quantitative analysis, the counterpart to reliability is conclusion validity. For more details, see Sect. 9.7.

It is, as described above, important to consider the validity of the case study from the beginning. Examples of ways to improve validity are triangulation; developing and maintaining a detailed case study protocol; having designs, protocols, etc. reviewed by peer researchers; having collected data and obtained results reviewed by case subjects; spending sufficient time with the case, and giving sufficient concern to analysis of "negative cases", i.e., looking for theories that contradict your findings.

7.8 Reporting

An empirical study cannot be distinguished from its reporting. The report communicates the findings of the study, but is also the main source of information for judging the quality of the study. Reports may have different audiences, such as peer researchers, policymakers, research sponsors, and industry practitioners [281]. This

may lead to the need for writing different reports for different audiences. Here, we focus on reports with peer researchers as the main audience, i.e., journal or conference articles and possibly accompanying technical reports [32]. Guidelines for reporting software engineering case studies to other audiences and in other formats are provided by Runeson et al. [212]. Benbasat et al. propose that due to the extensive amount of data generated in case studies, "books or monographs might be better vehicles to publish case study research" [32].

For case studies, the same high-level structure may be used as for experiments (see Chap. 12), but since case studies are more flexible and often mostly based on qualitative data, the low-level detail is often less standardized, but it may be very dependent on the individual case. Below, we first discuss the characteristics of a case study report and then a proposed structure.

7.8.1 Characteristics

Robson defines a set of characteristics that a case study report should have [207], which in summary implies that it should:

- Tell what the study was about.
- Communicate a clear sense of the studied case.
- Provide a "history of the inquiry" so the reader can see what was done, by whom, and how.
- Provide basic data in focused form, so the reader can make sure that the conclusions are reasonable.
- Articulate the researcher's conclusions and set them into a context they affect.

In addition, this must take place under the balance between the researcher's duty and goal to publish their results, and the companies' and individuals' integrity [4, 167].

Reporting the case study objectives and research questions is quite straightforward. If they are changed substantially over the course of the study, this should be reported to help in understanding the case.

Describing the case might be more sensitive, since this might enable identification of the case or its subjects. For example, "a large telecommunications company in Sweden" is most probably a branch of the Ericsson Corporation. However, the case may be better characterized by other means than only application domain and country. Internal characteristics, like size of the studied unit, average age of the personnel, etc. may be more interesting than external characteristics like domain and turnover. Either the case constitutes a small sub-unit of a large corporation, and then it can hardly be identified among the many sub-units, or it is a small company and hence it is hard to identify it among many candidates. Still, care must be taken to find this balance.

Providing a "history of the inquiry" requires a level of substantially more detail than pure reporting of used methodologies, for example, "we launched a case study

using semi-structured interviews and archival analysis." Since the validity of the study is highly related to what is done, by whom, and how, the sequence of actions and roles acting in the study process must be reported. On the other hand, there is no room for every single detail of the case study conduct, and hence a balance must be found. Data is collected in abundance in a qualitative study, and the analysis has as its main focus to reduce and organize data to provide a chain of evidence for the conclusions. However, to establish trust in the study, the reader needs relevant snapshots from the data that support the conclusions. These snapshots may be in the form of, for example, citations (typical or special statements), pictures, or narratives with anonymized subjects. Furthermore, categories used in the data classification, leading to certain conclusions, may help the reader follow the chain of evidence.

Finally, the conclusions must be reported and set into a context of implications, for example, by forming theories. A case study cannot be generalized in the meaning of being representative of a population, but this is not the only way of achieving and transferring knowledge. Conclusions can be drawn without statistics, and they may be interpreted and related to other cases. Communicating research results in terms of theories is an underdeveloped practice in software engineering [100], as discussed in Sect. 3.3.

7.8.2 Structure

For the academic reporting of case studies, the linear-analytic structure (problem, related work, methods, analysis, and conclusions) is the most accepted structure. The high-level structure for reporting experiments in software engineering proposed by Jedlitschka and Pfahl [119] therefore also fits the purpose of case study reporting. However, some adaptations are needed, based on specific characteristics of case studies and other issues based on an evaluation conducted by Kitchenham et al. [140]. The resulting structure is presented in Table 7.4.

In a case study, the theory may constitute a framework for the analysis; hence, there are two kinds of related work: (a) earlier studies on the topic and (b) theories on which the current study is based. The design section corresponds to the case study protocol, i.e., it reports the planning of the case study including the measures taken to ensure the validity of the study. Since the case study is of flexible design, and data collection and analysis are more intertwined, these topics may be combined into one section (as was done in Sect. 7.6).

Consequently, the contents at the lower level must be adjusted, as proposed in Table 7.4. Specifically for the combined data section, the coding scheme often constitutes a natural subsection structure. Alternatively, for a comparative case study, the data section may be structured according to the compared cases, and for a longitudinal study, the time scale may constitute the structure of the data section. This combined results section also includes an evaluation of the validity of the final results.

Table 7.4 Proposed reporting structure for case studies based on Jedlitschka and Pfahl [119] and adaptations to case study reporting according to Runeson et al [212]

Section headings	Subsections
Title	
Authorship	
Structured abstract	
Introduction	Problem statement
	Research objectives
	Context
Related work	Earlier studies
	Theory
Case study design	Research questions
	Case and subject selection
	Data collection procedure(s)
	Analysis procedure(s)
	Validity procedure(s)
Results	Case and subject descriptions, covering execution, analysis and interpretation issues
	Subsections, which may be structured, e.g., according to coding scheme, each linking observations to conclusions
	Evaluation of validity
Conclusions and future work	Summary of findings
	Relation to existing evidence
	Impact/implications
	Limitations
	Future work
Acknowledgments	
References	
Appendices	

Next, we turn to Part II of the book, where the steps in the experiment process are detailed in Chaps. 8–12.

7.9 Exercises

7.1 When is case study a feasible research methodology?

7.2 What is the purpose of planning in case studies, being a flexible research methodology?

7.3 Which criteria govern the selection of cases for a study?

7.4 List three types of interviews, and explain which type is suitable for different situations.

7.5 Describe a typical process for qualitative analysis.

Part II
Steps in the Experiment Process

Chapter 8
Scoping

Conducting an experiment is a labor-intensive task. In order to utilize the effort spent, it is important to ensure that the intention with the experiment can be fulfilled through the experiment. In the scoping step the foundation of the experiment is determined, which is illustrated in Fig. 8.1. If the foundation is not properly laid, rework may be required, or even worse, the experiment cannot be used to study what was intended. The purpose of the scoping step is to define the goals of an experiment according to a defined framework. Here we follow the GQM template for goal definition, originally presented by Basili and Rombach [21].

The scoping of an experiment is discussed in Sect. 8.1. An experiment goal definition example is presented in Sect. 8.2.

8.1 Scope Experiment

The scope of the experiment is set by defining its goals. The purpose of a goal definition template is to ensure that important aspects of an experiment are defined before the planning and execution take place. By defining the goal of the experiment according to this template, the foundation is properly laid.

The goal template includes the following [21]:

Analyze <Object(s) of study>
for the purpose of <Purpose>
with respect to their <Quality focus>
from the point of view of the <Perspective>
in the context of <Context>.

- The *object of study* is the entity that is studied in the experiment. The object of study can be products, processes, resources, models, metrics, or theories.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_8

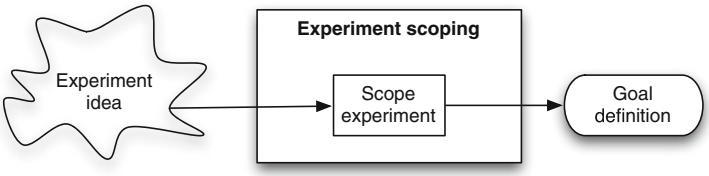


Fig. 8.1 Scoping step overview

Table 8.1 Experiment context classification

		# Objects	
		One	**More than one**
# Subjects per object	**One**	Single-object study	Multi-object variation study
	More than one	Multi-test within object study	Blocked subject-object study

Examples are the final product, the development or inspection process, and a reliability growth model.

- The *purpose* defines the intention of the experiment. It may be to evaluate the impact of two different techniques, or to characterize the learning curve of an organization.
- The *quality focus* is the primary effect under study in the experiment. Quality focus may be effectiveness, cost, reliability, etc.
- The *perspective* is the viewpoint from which the experiment results are interpreted. Examples of perspectives are developer, project manager, customer, and researcher.
- The *context* is the "environment" in which the experiment is run. The context briefly defines which personnel is involved in the experiment (subjects) and which software artifacts (objects[1]) are used in the experiment. Subjects can be characterized by experience, team size, workload, etc. Objects can be characterized by size, complexity, priority, application domain, etc. The experiment context can be classified in terms of the number of subjects and objects involved in the study [19] (see Table 8.1).

Single-object studies are conducted on a single subject and a single object. Multi-object variation studies are conducted on a single subject across a set of objects. Multi-test within object studies examine a single object across a set of subjects. Blocked subject-object studies examine a set of subjects and a set of objects. All these experiment types can be run either as an experiment or as a quasi-experiment. In a quasi-experiment there is a lack of randomization of either subjects or objects. The single-object study is a quasi-experiment if the single subject and object are not selected by random, but it is an experiment if the subject and object are chosen

[1] Note that the "objects" here are generally different from the "objects of study" defined above.

Table 8.2 Example experiment context classification, from Basili [19]

		# Objects	
		One	**More than one**
# Subjects per object	**One**	3. Cleanroom project no. 1 at SEL [20]	4. Cleanroom projects no. 2-4 at SEL [20]
	More than one	2. Cleanroom experiment at University of Maryland [218]	1. Reading versus test [22] 5. Scenario-based reading vs. checklist [25]

by random. The difference between experiments and quasi-experiments is discussed further by Robson [207].

Examples of the different experiment types are given by the series of experiments conducted at NASA Software Engineering Laboratory (NASA-SEL) [19], aimed at evaluation of Cleanroom Software Engineering principles and techniques. Cleanroom is a collection of engineering methods and techniques assembled with the objective to produce high-quality software. A brief introduction to Cleanroom is provided by Linger [159] and Mills et al. [177]. Cleanroom was primarily practiced before the turn of the millennium. However, the series of experiments provides an excellent illustration of the different contextual settings in Table 8.2. The experiment series consists of four distinct steps. First, a reading versus unit test experiment was conducted in a blocked subject-object study [22] (see 1 in Table 8.2). Second, a development project applying Cleanroom techniques was conducted in a student environment [218]. The experiment was a multi-test within object variation experiment (see 2 in Table 8.2). Third, a project using Cleanroom was conducted at NASA-SEL [20] as a single-object experiment (see 3 in Table 8.2). Fourth, three Cleanroom projects were conducted in the same environment, constituting a multi-object variation study [20] (see 4 in Table 8.2). The next round is a new reading experiment where different techniques are analyzed [25] (see 5 in Table 8.2). This series of experiments is also discussed by Linkman and Rombach [160].

The example in Table 8.2 illustrates how experiments (see 1 and 2) can be conducted as pre-studies prior to case studies (see 3 and 4). This is in line with the discussion regarding technology transfer and a suitable ordering based on cost and risk as discussed in Sects. 3.5 and 3.6.

8.2 Example Experiment

The goal definition framework can be filled out with different objects of study, purposes, etc. In Table 8.3, examples of elements are given.

A study definition example is constructed by composing the elements of the framework and is presented below. The example defines an inspection experiment where different inspection techniques are evaluated, i.e., perspective-based reading vs. checklist-based reading. Perspective-based reading was introduced by Basili et

Table 8.3 Goal definition framework

Object of study	Purpose	Quality focus	Perspective	Context
Product	Characterize	Effectiveness	Developer	Subjects
Process	Monitor	Cost	Modifier	Objects
Model	Evaluate	Reliability	Maintainer	
Metric	Predict	Maintainability	Project manager	
Theory	Control	Portability	Corporate manager	
	Change		Customer	
			User	
			Researcher	

al. [25], and it has been evaluated in several experiments including a comparison of perspective-based reading vs. an existing method at NASA by Maldonado et al. [164]. Laitenberger et al. [152] present a comparison between perspective-based reading and a checklist-based approach. Researchers have also compared other reading techniques, such as a comparison between usage-based reading and checklist-based reading by Thelin et al. [250].

The *objects studied* are the Perspective-Based Reading (PBR) technique and a checklist-based technique. The *purpose* is to evaluate the reading techniques, in particular with respect to differences between perspectives in PBR. The *quality focus* is the effectiveness and efficiency of the reading techniques. Effectiveness refers to the number of faults found of the total number of faults and efficiency refers to the number of faults found per time unit. The *perspective* is from the researcher's point of view. The experiment, which sets the *context*, is run using MSc and PhD students as subjects based on a defined lab package with textual requirements documents. The study is conducted as a blocked subject-object study (see Table 8.1), since it involves many subjects and more than one requirements document.

The example is summarized as follows:

Analyze *the PBR and checklist techniques*
for the purpose of *evaluation*
with respect to *effectiveness and efficiency*
from the point of view of *the researcher*
in the context of *MSc and PhD students' reading requirements documents*.

This example is used in Chaps. 9–11 to illustrate the progress of the experimental process. The summary of the experiment forms the goal definition of the experiment. It is the input to the planning step in the experiment process.

8.3 Exercises

8.1 Why is it important to have set up clear goals with an experiment from the beginning?

8.2 Write an example of a goal definition for an experiment you would like to conduct.

8.3 Why is the context in an experiment important?

8.4 How can the context be characterized?

8.5 Explain how a series of studies can be used for technology transfer from research to practice.

Chapter 9
Planning

After the scoping of the experiment, the planning takes place. The scoping determines the foundation for the experiment—*why* the experiment is conducted—while the planning prepares for *how* the experiment is conducted.

As in all types of engineering activities, the experiment must be planned and the plans must be followed up to control the experiment. The result of the experiment can be affected, or even rendered invalid, if not planned properly.

The planning step of an experiment can be divided into seven sub-steps. The input to the step is the goal definition for the experiment (see Chap. 8). Based on the goal definition, in the *context selection* sub-step, the environment in which the experiment will be executed is determined. Next, the *hypothesis formulation* and the *variable selection* of independent and dependent variables take place. The *selection of subjects* is carried out. The *experiment design type* is chosen based on the hypothesis and variables selected. Next the *instrumentation* prepares for the practical implementation of the experiment. Finally the *validity evaluation* aims at checking the validity of the experiment. The planning process is iterated until a complete experiment design is ready. An overview of the planning step is given in Fig. 9.1. The different sub-steps in the planning step are presented in Sects. 9.1–9.7. These are complemented with a more detailed discussion of validity threats in Sect. 9.8 and priority among them in Sect. 9.9. The chapter concludes with an example experiment illustrating the sub-steps in the planning step in Sect. 9.10.

9.1 Context Selection

In order to achieve the most general results in an experiment, it should be executed in large, real software projects, with professional staff. However, conducting an experiment involves risks, for example, that the new method to be examined is not as good as expected and causes delays. An alternative is to run offline projects

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_9

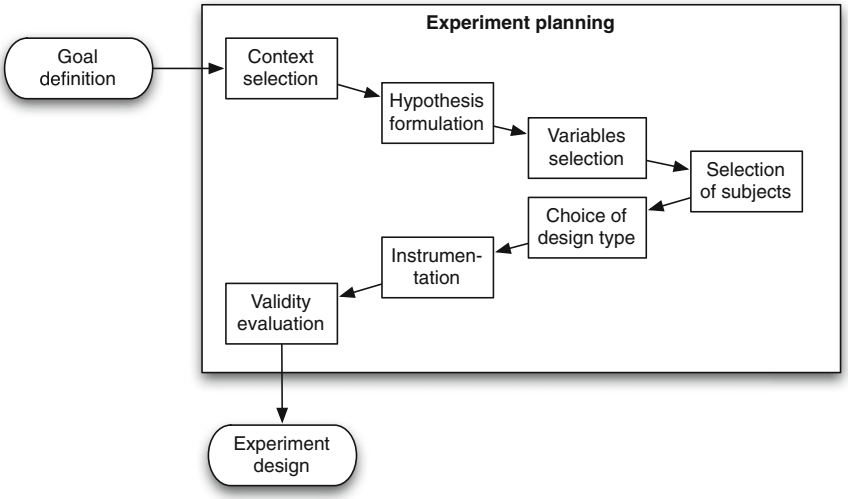


Fig. 9.1 Planning step overview

in parallel with the real projects. This reduces the risks but causes extra costs. A cheaper alternative is to run projects staffed by students. Such projects are cheaper and easier to control, but more directed to a certain context than projects staffed by professionals with more and various types of experience. Furthermore, these projects seldom address real problems, but problems more of toy size due to constraints in cost and time. This trade-off involves a balance between making studies valid to a specific context or valid to the general software engineering domain. This is further elaborated in Sect. 9.7. Given this trade-off, experiments with students as subjects are discussed in literature, for example, by Höst et al. [109] and Falessi et al. [73].

Hence, the context of the experiment can be characterized according to four dimensions:

- Offline vs. online or real life
- Student vs. professional
- Toy vs. real problems
- Specific vs. general

A common situation in an experiment is that something existing is compared to something new; for example, an existing inspection method is compared to a new one [25, 198, 201]. There are two problems related to these types of studies. First, what is the existing method? It has been applied for some period of time, but it is rarely well documented and there is no consistent application of the method. Second, learning a new method may influence how the old one is applied.

This and other human-related issues need to be considered when planning for an experiment to make the results valid.

9.2 Hypothesis Formulation

The basis for statistical analysis of an experiment is hypothesis testing. A hypothesis is stated formally and the data collected during the course of the experiment is used to, if possible, reject the hypothesis. If the hypothesis can be rejected then conclusions can be drawn, based on the hypothesis testing under given risks. It is essential to note that if the hypothesis cannot be rejected, it does *not* imply that the hypothesis can be accepted. Potentially the hypothesis could have been rejected with more data points. Thus, a hypothesis (or the null hypothesis; see below) can never be accepted, only rejected.

In the planning step, the experiment definition is formalized into hypotheses. Two hypotheses have to be formulated:

Null — A null hypothesis, H_0, states that there are no real underlying trends or patterns in the experiment setting; the only reasons for differences in our observations are coincidental. This is the hypothesis that the experimenter wants to reject with as high significance as possible. An example hypothesis is that a new inspection method finds on average the same number of faults as the old one, i.e., $H_0 : \mu_{N\,old} = \mu_{N\,new}$, where μ denotes the average and N is the number of faults found.

Alternative — An alternative hypothesis, H_a *or* H_1, is the hypothesis in favor of which the null hypothesis is rejected. An example hypothesis is that a new inspection method on average finds more faults than the old one, i.e., $H_1 : \mu_{N\,old} < \mu_{N\,new}$.

There are a number of different statistical tests described in the literature that can be used to evaluate the outcome of an experiment. They are all based on the fact that the above hypotheses are formulated before the statistical tests are chosen and performed. The statistical tests are elaborated in Sect. 11.3.

Testing hypotheses involves different types of risks. Either the test rejects a true hypothesis or the test does not reject a false hypothesis. These risks are referred to as type-I-error and type-II-error:

Type-I-error — A type-I-error has occurred when a statistical test has indicated a pattern or relationship even if there actually is no real pattern. That is, the probability of committing a type-I-error can be expressed as: $P(\text{type-I-error}) = P(\text{reject } H_0 \mid H_0 \text{ true})$.
In the example hypothesis above, the type-I-error is the probability of rejecting H_0 even though the two methods on average find the same number of faults.

Type-II-error — A type-II-error has occurred when a statistical test has not indicated a pattern or relationship even if there actually is a real pattern. That is, the probability of committing a type-II-error can be expressed as: $P(\text{type-II-error}) = P(\text{not reject } H_0 \mid H_0 \text{ false})$. In the example hypothesis above, the type-II-error is the probability of not rejecting H_0 even though the two methods on average have different means.

The size of the errors depends on different factors. One example is the ability of the statistical test to reveal a true pattern in the collected data. This is referred to as the power of a test:

Power The power of a statistical test is the probability that the test will reveal a true pattern if H_0 is false. An experimenter should choose a test with as high power as possible. The power can be expressed as:
Power $= P(\text{reject } H_0 \mid H_0 \text{ false}) = 1 - P(\text{type-II-error})$

All these factors have to be considered when planning an experiment.

9.3 Variables Selection

Before any design can start we have to choose the dependent and independent variables.

The *independent variables* are those variables that we can control and change in the experiment. Choosing the right variables is not easy and it usually requires domain knowledge. The variables should have some effect on the dependent variable and must be controllable. The choices of the independent and dependent variables are often made simultaneously or in reverse order. The choice of independent variables also includes choosing the measurement scales, the range for the variables, and the specific levels at which tests will be made.

The effect of the treatments is measured in the *dependent variable(s)*. Often there is only one dependent variable and it should therefore be derived directly from the hypothesis. The variable is generally not directly measurable and we have to measure it via an indirect measure instead. This indirect measure must be carefully validated, because it affects the result of the experiment. The hypothesis can be refined when we have chosen the dependent variable. The choice of dependent variable also means that the measurement scale and range of the variables are determined. A reason to have only one dependent variable is that if there are more, there is a risk that the "fishing and the error rate" threat to conclusion validity may become too large as described in Sect. 9.8.1.

9.4 Selection of Subjects

The selection of subjects is important when conducting an experiment [15, 207]. The selection is closely connected to the generalization of the results from the experiment. In order to generalize the results to the desired population, the selection must be representative for that population. The selection of subjects is also called a sample from a population.

The sampling of the population can be either a probability or a non-probability sample. The difference between the two is that in the probability sampling, the

probability of selecting each subject is known and in the non-probability sampling it is unknown. Examples of *probability sampling techniques* are as follows:

- *Simple random sampling:* Subjects are selected from a list of the population at random.
- *Systematic sampling:* The first subject is selected from the list of the population at random and then every n:th person is selected from the list.
- *Stratified random sampling:* The population is divided into a number of groups or strata with a known distribution between the groups. Random sampling is then applied within the strata.

Examples of *non-probability sampling techniques* are as follows:

- *Convenience sampling:* The nearest and most convenient persons are selected as subjects.
- *Quota sampling:* This type of sampling is used to get subjects from various elements of a population. Convenience sampling is normally used for each element.

The size of the sample also impacts the results when generalizing. The larger the sample, the lower the error becomes when generalizing the results. The sample size is also closely related to the power of the statistical test (see Sect. 11.3.1). There are some general principles for choosing the sample size:

- If there is large variability in the population, a larger sample size is needed.
- The analysis of the data may influence the choice of the sample size. It is therefore necessary to consider how the data shall be analyzed already at the design stage of the experiment.

9.5 Experiment Design

To draw meaningful conclusions from an experiment, we apply statistical analysis methods on the collected data to interpret the results, as further described in Chap. 11. To get the most out of the experiment, it must be carefully planned and designed. Which statistical analyses we can apply depends on the chosen design, and the measurement scales used (see Sect. 3.4). Therefore design and interpretation are closely related.

9.5.1 Choice of Experiment Design

An experiment consists of a series of tests of the treatments. A design of an experiment describes how the tests are organized and run. More formally, we can define an experiment as a set of tests.

As described above, the design and the statistical analysis are closely related. The choice of design affects the analysis and vice versa. To design the experiment, we have to look at the hypothesis to see which statistical analysis we have to perform to potentially reject the null hypothesis. Based on the statistical assumptions, for example, the measurement scales, and on which objects and subjects we are able to use, we make the experiment design. During the design we determine how many tests the experiment shall have to make sure that the effect of the treatment is visible. A proper design also forms the basis to allow for replication. In the following two sections, general design principles and some standard design types are presented.

9.5.2 General Design Principles

When designing an experiment, many aspects must be considered. The general design principles are *randomization, blocking*, and *balancing*, and most experiment designs use some combination of these. To illustrate the general design principles, we use an example.

Example A company will conduct an experiment to investigate the effect on the reliability of a program when using an object-oriented design instead of the standard company design principle. The experiment will use program A as the experiment object. The experiment design is of type "multi-test within object study" (see Chap. 8).

Randomization One of the most important design principles is randomization. All statistical methods used for analyzing the data require that the observations be from independent random variables. To meet this requirement, randomization is used. The randomization applies to the allocation of the objects and subjects and in which order the tests are performed. Randomization is used to average out the effect of a factor that may otherwise be present. It is also used to select subjects that are representative of the population of interest.

Example The selection of the persons (subjects) will be representative of the designers in the company, by random selection of the available designers. The assignment to each treatment (the object-oriented design or the standard company design principle) is selected randomly.

Blocking Sometimes we have a factor that probably has an effect on the response, but we are not interested in that effect. If the effect of the factor is known and controllable, we can use a design technique called blocking. Blocking is used to systematically eliminate the undesired effect in the comparison among the treatments. Within one block, the undesired effect is the same and we can study the effect of the treatments on that block. Blocking is used to eliminate the undesired effect in the study and therefore the effects between the blocks are not studied. This technique increases the precision of the experiment.

Example The persons (subjects) used for this experiment have different experiences. Some of them have used object-oriented design before and some have not. To minimize the effect of the experience, the persons are grouped into two groups (blocks), one with experience of object-oriented design and one without.

Balancing If we assign the treatments so that each treatment has an equal number of subjects, we have a balanced design. Balancing is desirable because it both simplifies and strengthens the statistical analysis of the data, but it is not necessary.

Example The experiment uses a balanced design, which means that there is the same number of persons in each group (block).

9.5.3 Standard Design Types

In this section some of the most frequently used experiment designs are presented. The designs range from simple experiments with a single factor to more complex experiments with many factors. Experiment design is discussed in depth by, for example, Montgomery [180] and is elaborated in more depth for software engineering by Juristo and Moreno [122]. For most of the designs, an example hypothesis is formulated and statistical analysis methods are suggested for each design. The design types presented in this section are suitable for experiments with the following:

- One factor with two treatments
- One factor with more than two treatments
- Two factors with two treatments
- More than two factors each with two treatments

One Factor with Two Treatments With these experiments, we want to compare the two treatments against each other. The most common is to compare the means of the dependent variable for each treatment.

The following notations are used:

μ_i The mean of the dependent variable for treatment i
y_{ij} The j:th measure of the dependent variable for treatment i

Example of an Experiment The aim is to investigate if a new design method produces software with higher quality than the previously used design method. The factor in this experiment is the design method and the treatments are the new and the old design methods. The dependent variable can be the number of faults found in development.

Completely Randomized Design This is a basic experiment design for comparing two treatment means. The design setup uses the same objects for both treatments and assigns the subjects randomly to each treatment (see Table 9.1). Each subject

Table 9.1 Example of assigning subjects to the treatments for a randomized design

Subjects	Treatment 1	Treatment 2
1	X	
2		X
3		X
4	X	
5		X
6	X	

uses only one treatment on one object. If we have the same number of subjects per treatment the design is balanced.

Examples of hypothesis:

$H_0 : \mu_1 = \mu_2$
$H_1 : \mu_1 \neq \mu_2,\ \mu_1 < \mu_2$ or $\mu_1 > \mu_2$

Examples of analysis: t-test and Mann-Whitney (see Sect. 11.3).

Paired Comparison Design The fact that different subjects perform differently in an experiment contributes to the error in the experiment, which is typically handled by randomization. This error can also be reduced by "pairing" the design, i.e., letting every subject in a two-factor experiment be subject to both treatments. That is, we can improve the precision of the experiment by making comparisons within matched pairs of experiment material. In this design, each subject uses both treatments on the same object. This is sometimes referred to as a crossover design or repeated measure design.

There are obvious challenges that need to be solved with this type of design. In a medical experiment investigating the effect of two different treatments, a problem would be if the effect of one of the treatments remains when the effects of the other are measured. The same problem can be seen if, for example, two inspection or review methods are used and the subjects remember the comments from using the first method when using the second method. These challenges are further discussed in relation to the example in Sect. 11.4.

One solution is to use different, but similar, experiment material in the trials for each subject. For example, if a requirements specification is reviewed, similarly structured specifications describing different functionality could be used. Then, the next problem will be that the different objects that are used are not as identical as the researcher hopes, and that the order may affect the result. To minimize the effect of the order in which the subjects apply the treatments, the order is assigned randomly to each subject (see Table 9.2). The analysis of the experiment with this design will be to see if the difference between the paired measures is zero.

If we have the same number of subjects starting with the first treatment as with the second, we have a balanced design. Paired designs can be applied to more than two-factor experiments. See, for example, "Completely randomized design" below.

Table 9.2 Example of assigning the treatments for a paired design

Subjects	Treatment 1	Treatment 2
1	2	1
2	1	2
3	2	1
4	2	1
5	1	2
6	1	2

Table 9.3 Example of assigning the treatments to the subjects

Subjects	Treatment 1	Treatment 2	Treatment 3
1		X	
2			X
3	X		
4	X		
5		X	
6			X

Examples of hypothesis:

$d_j = y_{1j} - y_{2j}$ and μ_d is the mean of the difference.
$H_0 : \mu_d = 0$
$H_1 : \mu_d \neq 0,\ \mu_d < 0$ or $\mu_d > 0$

Examples of analysis: Paired t-test, Sign test, and Wilcoxon (see Sect. 11.3).

One Factor with More than Two Treatments As with experiments with only two treatments, we want to compare the treatments with each other. The comparison is often performed on the treatment means.

Example of an Experiment The experiment investigates the quality of the software when using different programming languages. The factor in the experiment is the programming language and the treatments can be C, C++, and Java.

Completely Randomized Design A completely randomized design requires that the experiment is performed in random order so that the treatments are used in as uniform an environment as possible. The design uses one object to all treatments and the subjects are assigned randomly to the treatments (see Table 9.3).

Examples of hypothesis, where a is the number of treatments:

$H_0 : \mu_1 = \mu_2 = \mu_3 = \ldots = \mu_a$
$H_1 : \mu_i \neq \mu_j$ for at least one pair (i, j)

Examples of analysis: ANOVA (ANalysis Of VAriance) and Kruskal-Wallis (see Sect. 11.3).

Randomized Complete Block Design If the variability between the subjects is large, we can minimize this effect on the result by using a randomized complete block design. With this design, each subject uses all treatments and the subjects form a

Table 9.4 Example of assigning the treatments to the subjects

Subjects	Treatment 1	Treatment 2	Treatment 3
1	1	3	2
2	3	1	2
3	2	3	1
4	2	1	3
5	3	2	1
6	1	2	3

Table 9.5 Example of a 2*2 factorial design

		Factor A	
		Treatment A1	**Treatment A2**
Factor B	**Treatment B1**	Subject 4, 6	Subject 1, 7
	Treatment B2	Subject 2, 3	Subject 5, 8

more homogeneous experiment unit, i.e., we block the experiment on the subjects (see Table 9.4). The blocks represent a restriction on randomization. The experiment design uses one object to all treatments and the order in which the subjects use the treatments is assigned randomly. The paired comparison design above is a special case of this design with only two treatments. The randomized complete block design is one of the most used experiment designs.

Examples of hypothesis:

$H_0 : \mu_1 = \mu_2 = \mu_3 = \ldots = \mu_a$
$H_1 : \mu_i \neq \mu_j$ for at least one pair (i, j)

Examples of analysis: ANOVA (ANalysis Of VAriance) and Kruskal-Wallis (see Sect. 11.3).

Two Factors The experiment gets more complex when we increase from one factor to two. The single hypothesis for the experiments with one factor will split into three hypotheses: one hypothesis for the effect from one of the factors, one for the other, and one for the interaction between the two factors. We use the following notations:

τ_i The effect of treatment i on factor A
β_j The effect of treatment j on factor B
$(\tau\beta)_{ij}$ The effect of the interaction between τ_i and β_j

$2 * 2$ *Factorial Design* This design has two factors, each with two treatments. In this experiment design, we randomly assign subjects to each combination of the treatments (see Table 9.5).

Example of an Experiment The experiment investigates the understandability of the design document when using structured or object-oriented design based on one "good" and one "bad" requirements documents. The first factor, A, is the design method and the second factor, B, is the requirements document. The experiment design is a 2*2 factorial design as both factors have two treatments and every combination of the treatments is possible.

Table 9.6 Example of a two-stage nested design where B is nested under A

Factor A			
Treatment A1		Treatment A2	
Factor B		Factor B	
Treatment B1'	Treatment B2'	Treatment B1"	Treatment B2"
Subject 1, 3	Subject 6, 2	Subject 7, 8	Subject 5, 4

Examples of hypothesis:

$H_0 : \tau_1 = \tau_2 = 0$
H_1 : at least one $\tau_i \neq 0$
$H_0 : \beta_1 = \beta_2 = 0$
H_1 : at least one $\beta_j \neq 0$
$H_0 : (\tau\beta)_{ij} = 0$ for all i, j
H_1 : at least one $(\tau\beta)_{ij} \neq 0$

Example of analysis: ANOVA (ANalysis Of VAriance) (see Sect. 11.3).

Two-Stage Nested Design If one of the factors, for example B, in the experiment is similar but not identical for different treatments of the other factor, for example A, we have a design that is called nested or hierarchical design. Factor B is said to be nested under factor A. The two-stage nested design has two factors, each with two or more treatments. The experiment design and analysis are the same as for the 2*2 factorial design (see Table 9.6).

Example of an Experiment The experiment investigates the test efficiency of unit testing of a program when using function or object-oriented programming and if the programs are "defect-prone" or "non-defect-prone." The first factor, A, is the programming language and the second factor, B, is the defect-proneness of the program. The experiment design has to be nested, as a "defect-prone/non-defect-prone" functional program is not the same as a "defect-prone/non-defect-prone" object-oriented program.

More than Two Factors In many cases, the experiment has to consider more than two factors. The effect in the dependent variable can therefore be dependent not only on each factor separately but also on the interactions between the factors. These interactions can be between two or more factors. This type of design is called a factorial design. This section gives an introduction to designs where each factor has only two treatments. Designs where the factors have more than two treatments are presented by Montgomery [180].

2^k *Factorial Design* The 2*2 factorial design is a special case of the 2^k factorial design, i.e., when $k = 2$. The 2^k factorial design has k factors where each factor has two treatments. This means that there are 2^k different combinations of the treatments. To evaluate the effects of the k factors, all combinations have to

Table 9.7 Example of a 2^3 factorial design

Factor A	Factor B	Factor C	Subjects
A1	B1	C1	2, 3
A2	B1	C1	1, 13
A1	B2	C1	5, 6
A2	B2	C1	10, 16
A1	B1	C2	7, 15
A2	B1	C2	8, 11
A1	B2	C2	4, 9
A2	B2	C2	12, 14

be tested. The subjects are randomly assigned to the different combinations. An example of a 2^3 factorial design is shown in Table 9.7.

The hypotheses and the analyses for this type of design are of the same type as for the 2*2 factorial design. More details about the 2^k factorial design are presented by Montgomery [180].

2^k *Fractional Factorial Design* When the number of factors grows in a 2^k factorial design, the number of factor combinations grows rapidly; for example, there are 8 combinations for a 2^3 factorial design and 16 for a 2^4 factorial design. Often, it can be assumed that the effects of certain high-order interactions are negligible and that the main effects and the low-order interaction effects can be obtained by running a fraction of the complete factorial experiment. This type of design is therefore called fractional factorial design.

The fractional factorial design is based on three ideas:

- *The sparsity of effect principle:* It is likely that the system is primarily driven by some of the main and low-order interaction effects.
- *The projection property:* A stronger design can be obtained by taking a subset of significant factors from the fractional factorial design.
- *Sequential experimentation:* A stronger design can be obtained by combining sequential runs of two or more fractional factorial designs.

The major use of these fractional factorial designs is in screening experiments, where the purpose of the experiment is to identify the factors that have large effects on the system. Examples of fractional factorial designs are as follows:

One-Half Fractional Factorial Design of the 2^k Factorial Design Half of the combinations of a full 2^k factorial design is chosen. The combinations are selected so that if one factor is removed the remaining design is a full 2^{k-1} factorial design(see Table 9.8). The subjects are randomly assigned to the selected combinations. There are two alternative fractions in this design and if both fractions are used in sequence, the resulting design is a full 2^k factorial design.

Table 9.8 Example of a one-half fraction of the 2^3 factorial design

Factor A	Factor B	Factor C	Subjects
A1	B1	C2	2, 3
A2	B1	C1	1, 8
A1	B2	C1	5, 6
A2	B2	C2	4, 7

Table 9.9 Example of a one-quarter fraction of the 2^5 factorial design

Factor A	Factor B	Factor C	Factor D	Factor E	Subjects
A1	B1	C1	D2	E2	3, 16
A2	B1	C1	D1	E1	7, 9
A1	B2	C1	D1	E2	1, 4
A2	B2	C1	D2	E1	8, 10
A1	B1	C2	D2	E1	5, 12
A2	B1	C2	D1	E2	2, 6
A1	B2	C2	D1	E1	11, 15
A2	B2	C2	D2	E2	13, 14

One-Quarter Fractional Factorial Design of the 2^k Factorial Design One quarter of the combinations of the full 2^k factorial design is chosen. The combinations are selected so that if two factors are removed the remaining design is a full 2^{k-2} factorial design (see Table 9.9). There are, however, dependencies between the factors in the one-quarter design due to the fact that it is not a full factorial design.

For example, in Table 9.9, factor D is dependent on a combination of factors A and B. It can, for example, be seen that for all combinations of A1 and B1, we have D2, and so forth. In a similar way, factor E is dependent on a combination of factors A and C. Thus, if factors C and E (or B and D) are removed, the resulting design becomes two replications of a 2^{3-1} fractional factorial design and not a 2^3 factorial design. The latter design is obtained if D and E are removed. The two replications can be identified in Table 9.9 by noticing that the first four rows are equivalent to the four last rows in the table, when C and E are removed, and hence it becomes two replications of a 2^2 factorial design.

The subjects are randomly assigned to the selected combinations. There are four alternative fractions in this design and if all four fractions are used in sequence, the resulting design is a full 2^k factorial design. If two of the fractions are used in sequence a one-half fractional design is achieved.

More details on fractional factorial designs are presented by Montgomery [180].

In summary, the choice of the correct experimental design is crucial, since a poor design will undoubtedly affect the possibility of being able to draw the correct conclusions after the study. Furthermore, the design puts constraints on the statistical methods that can be applied. Finally, it should be stressed that it is important to try to use a simple design if possible and to make the best possible use of the available subjects.

9.6 Instrumentation

The instruments for an experiment are of three types, namely objects, guidelines, and measurement instruments. The instruments are chosen in the planning of an experiment. Before execution, the instruments are developed for the specific experiment.

Experiment objects may be, for example, specification or code documents. When planning for an experiment, it is important to choose objects that are appropriate. For example, in an inspection experiment, the number of faults must be known in the inspection objects. This can be achieved by seeding faults or by using a document with a known number of faults. Using a true early version of a document in which the faults are identified can do the latter.

Guidelines are needed to guide the participants in the experiment. Guidelines include, for example, process descriptions and checklists. If different methods are compared in the experiment, guidelines for the methods have to be prepared for the experiment. It is essential that the guidelines provide comparable support to ensure that the guidelines do not affect the results, i.e., the guidelines should not become a confounding factor. As mentioned in the context of case study research (see Chap. 7), a confounding factor makes it impossible to distinguish the effects of two factors from each other. In addition to the guidelines, the participants also need training in the methods to be used.

Measurements in an experiment are conducted via data collection. In human-intensive experiments, data is generally collected via manual forms or in interviews. The planning task to be performed is to prepare forms and interview questions and to validate the forms and questions with some people having similar background and skills as the experiment participants. An example of a form used to collect information about the experience of subjects is shown among the exercises (see Table A.1 in Appendix A).

The overall goal of the instrumentation is to provide means for performing the experiment and to monitor it, without affecting the control of the experiment. The results of the experiment shall be the same independently of how the experiment is instrumented. If the instrumentation affects the outcome of the experiment, the results are invalid.

The validity of an experiment is elaborated in Sect. 9.7 and more about the preparation of instruments can be found in Sects. 10.1.2 and 10.2.2.

9.7 Validity Evaluation

A fundamental question concerning results from an experiment is how valid they are. It is important to consider the question of validity already in the planning step to plan for adequate validity of the experiment results [119, 259]. Adequate validity refers to the fact that the results should be valid for the population of interest. First

of all, the results should be valid for the population from which the sample is drawn. Second, it may be of interest to generalize the results to a broader population. The results are said to have adequate validity if they are valid for the population to which we would like to generalize.

Adequate validity does not necessarily imply most general validity. An experiment conducted within an organization may be designed to answer some questions for that organization exclusively, and it is sufficient if the results are valid within that specific organization. A/B testing experiments are typical examples where the validity for a specific organization and their users is the primary goal [209]. On the other hand, if more general conclusions shall be drawn, the validity must cover a more general scope as well.

Baltes and Ralph [15] analyzed sampling methods in 115 empirical studies, published in four major software engineering venues, between 2014 and 2019. They found that most of them applied purposive sampling (73.0%) or convenience sampling (11.3%), while only a few used random sampling (6.4%) or stratified random sampling (2.0%). For interpretivist studies, this is not critical, while "Sampling is crucial for positivist studies because it determines external validity" [15]. The practice of analyzing and reporting threats to validity in software engineering research is also discussed and criticized for being a superficial afterthought, rather than a critical reflection on the potential threats of studies [259].

There are different classification schemes for different types of threats to the validity of an experiment. Campbell and Stanley define two types, threats to internal and external validity [44]. Cook and Campbell extend the list to four types of threats to the validity of experimental results, namely *conclusion, internal, construct*, and *external validity* [50]. The former categorization is sometimes referred to in the literature, but the latter is preferable since it is easily mapped to the different steps involved when conducting an experiment. Each of the four categories presented by Cook and Campbell [50] is related to a methodological question in experimentation, as follows.

The basic principles of an experiment are presented in Fig. 9.2. On the top, we have the theory area, and on the bottom, the observation area. We want to draw conclusions about the theory defined in the hypotheses, based on our observations. In drawing conclusions we have four sub-steps, in each of which there is one type of threat to the validity of the results.

1. *Conclusion validity.* This validity is concerned with the relationship between the treatment and the outcome. We want to make sure that there is a statistical relationship, i.e., with a given significance.
2. *Internal validity.* If a relationship is observed between the treatment and the outcome, we must make sure that it is a causal relationship, and that it is not a result of a factor over which we have no control or have not measured. In other words, we must ensure that the treatment causes the outcome (the effect).
3. *Construct validity.* This validity is concerned with the relation between theory and observation. If the relationship between cause and effect is causal, we must ensure two things: (1) that the treatment reflects the construct of the cause well

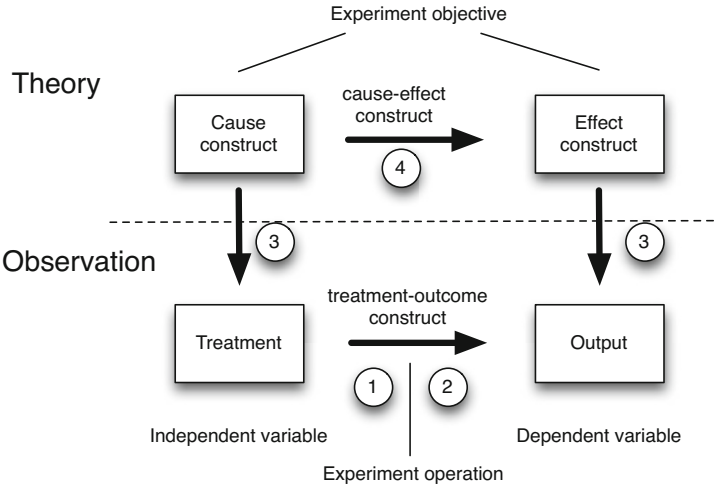


Fig. 9.2 Experiment principles (adapted from Trochim [253])

(see left part of Fig. 9.2) and (2) that the outcome reflects the construct of the effect well (see right part of Fig. 9.2) [233].

4. *External validity.* External validity is primarily concerned with generalization. If there is a causal relationship between the construct of the cause and the effect, can the result of the study be generalized outside the scope of our study? Is there a relation between the treatment and the outcome?

Conclusion validity is sometimes referred to as statistical conclusion validity [50], and has its counterpart in reliability for qualitative analysis (see Sect. 7.7.3). Threats to conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment. These issues include, for example, choice of statistical tests, choice of sample sizes, and care taken in the implementation and measurement of an experiment.

Threats to internal validity concern issues that may indicate a causal relationship, although there is none. Factors that impact on internal validity are how the subjects are selected and divided into different classes, how the subjects are treated and compensated during the experiment, if special events occur during the experiment, etc. All these factors can make the experiment show a behavior that is not due to the treatment but to a confounding factor.

Threats to construct validity refer to the extent to which the experiment setting and measurements actually reflect the construct under study. For example, the number of courses taken at the university in computer science may be a poor measure of the subject's experience in a programming language, i.e., has poor construct validity. The number of years of practical use may be a better measure, i.e., has better construct validity.

Threats to external validity concern the ability to generalize experiment results outside the experiment setting. External validity is affected by the experiment design chosen, but also by the objects in the experiment and the subjects chosen. There are three main risks: having wrong participants as subjects, conducting the experiment in the wrong environment, and performing it with a timing that affects the results.

A detailed list of threats to the validity is presented in Sect. 9.8. This list can be used as a checklist for an experiment design, although the validity analysis also must include creative thinking to find validity threats of the specific experiment. In the validity evaluation, each of the items can be checked to trigger the analysis of threats for the specific experiments. Identified threats have to be addressed or accepted, since sometimes some threat to validity has to be accepted. It may even be impossible to carry out an experiment without certain threats, and hence they have to be accepted and then addressed when interpreting the results. In either case, this should be reported as a part of the study design. The priority between different types of threats is further discussed in Sect. 9.9.

9.8 Detailed Description of Validity Threats

Below, a list of threats to the validity of experiments is discussed based on Cook and Campbell [50]. All threats are not applicable to all experiments, but this list can be seen as a checklist. The threats are summarized in Table 9.10 and the alternative shorter version of the classification scheme [44] is summarized in Table 9.11.

9.8.1 Conclusion Validity

Threats to conclusion validity are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of an experiment.

Low Statistical Power The power of a statistical test is the ability of the test to reveal a true pattern in the data. If the power is low, there is a high risk that an erroneous conclusion will be drawn (for further details see Sect. 9.2, or more specifically we are unable to reject an erroneous hypothesis. This is a common threat to software engineering experiments, as observed by Dybå et al. [64].

Violated Assumptions of Statistical Tests Certain tests have assumptions on, for example, normally distributed data and independent samples. Violating the assumptions may lead to the wrong conclusions. Some statistical tests are more robust to violated assumptions than others (see Chap. 11).

Table 9.10 Threats to validity according to Cook and Campbell [50]

Conclusion validity	**Internal validity**
Low statistical power	History
Violated assumption of statistical tests	Maturation
Fishing and the error rate	Testing
Reliability of measures	Instrumentation
Reliability of treatment implementation	Statistical regression
Random irrelevancies in experimental setting	Selection
Random heterogeneity of subjects	Mortality
	Ambiguity about direction of causal influence
	Interactions with selection
	Diffusion of imitation of treatments
	Compensatory equalization of treatments
	Compensatory rivalry
	Resentful demoralization
Construct validity	**External validity**
Inadequate pre-operational explication of constructs	Interaction of selection and treatment
Mono-operation bias	Interaction of setting and treatment
Mono-method bias	Interaction of history and treatment
Confounding constructs and levels of constructs	
Interaction of different treatments	
Interaction of testing and treatment	
Restricted generalizability across constructs	
Hypothesis guessing	
Evaluation apprehension	
Experimenter expectancy	

Table 9.11 Threats to validity according to Campbell and Stanley [44]

Internal validity	**External validity**
History	Interaction of selection and treatment
Maturation	Interaction of history and treatment
Testing	Interaction of setting and treatment
Instrumentation	Interaction of different treatments
Statistical regression	
Selection	

Fishing and the Error Rate This threat contains two separate parts. Searching or "fishing" for a specific result is a threat, since the analyses are no longer independent and the researchers may influence the result by looking for a specific outcome. The

error rate is concerned with the actual significance level. For example, conducting three investigations with a significance level of 0.05 means that the total significance level is $1 - (1 - 0.05)^3$, which equals 0.14. The error rate (i.e., significance level) should thus be adjusted when conducting multiple analyses.

Reliability of Measures The validity of an experiment is highly dependent on the reliability of the measures. This in turn may depend on many different factors, like poor question wording, infeasible metrics, bad instrumentation, or bad instrument layout. The basic principle is that when you measure a phenomenon twice, the outcome shall be the same. For example, lines of code are more reliable than function points since they do not involve human judgment. In other words, objective measures that can be repeated with the same outcome are more reliable than subjective measures (see also Sect. 3.4).

Reliability of Treatment Implementation The implementation of the treatment means the application of treatments to subjects. There is a risk that the implementation will not be similar between different persons applying the treatment or between different occasions. The implementation should hence be as standardized as possible over different subjects and occasions.

Random Irrelevancies in Experimental Setting Elements outside the experimental setting may disturb the results, such as noise outside the room or a sudden interruption to the experiment.

Random Heterogeneity of Subjects There is always heterogeneity in a study group. If the group is very heterogeneous, there is a risk that the variation due to individual differences will be larger than due to the treatment. Choosing more homogeneous groups will on the other hand affect external validity (see below). For example, an experiment with undergraduate students reduces the heterogeneity, since they have more similar knowledge and background, but also reduces the external validity of the experiment, since the subjects are not selected from a general enough population.

9.8.2 Internal Validity

Threats to internal validity are influences that can affect the dependent variable with respect to causality, without the researcher's knowledge. Thus they threaten the conclusion about a possible causal relationship between treatment and outcome. The internal validity threats are sometimes sorted into three categories, *single group threats, multiple group threats*, and *social threats*.

Single Group Threats These threats apply to experiments with single groups. We have no control group to which we do not apply the treatment. Hence, there are problems in determining if the treatment or another factor caused the observed effect.

History In an experiment, different treatments may be applied to the same object at different times. Then there is a risk that the history will affect the experimental results, since the circumstances are not the same on both occasions, for example, if one of the experiment occasions is on the first day after a holiday or on a day when a very rare event takes place, and the other occasion is on a normal day.

Maturation This is the effect of the fact that the subjects react differently as time passes. Examples are when the subjects are affected negatively (tired or bored) during the experiment, or positively (learning) during the course of the experiment.

Testing If the test is repeated, the subjects may respond differently at different times since they know how the test is conducted. If there is a need for familiarization to the tests, it is important that the results of the test are not fed back to the subject, i.e., to not support unintended learning.

Instrumentation This is the effect caused by the artifacts used for experiment execution, such as data collection forms, document to be inspected in an inspection experiment, etc. If these are badly designed, the experiment is affected negatively.

Statistical Regression This is a threat when the subjects are classified into experimental groups based on a previous experiment or case study, for example top-10 or bottom-10. In this case there might be an increase or improvement, even if no treatment is applied at all. For example, if the bottom-10 in an experiment are selected as subjects based on a previous experiment, all of them will probably not be among the bottom-10 in the new experiment due to pure random variation. The bottom-10 cannot be worse than remaining among the bottom-10, and hence the only possible change is for the better, relatively the larger population from which they are selected.

Selection This is the effect of natural variation in human performance. Depending on how the subjects are selected from a larger group, the selection effects can vary. Furthermore, the effect of letting volunteers take part in an experiment may influence the results. Volunteers are generally more motivated and suited for a new task than the whole population. Hence the selected group is not representative of the whole population.

Mortality This effect is due to the different kinds of persons who drop out from the experiment. It is important to characterize the dropouts to check if they are representative of the total sample. If subjects of a specific category drop out, for example, all the senior reviewers in an inspection experiment, the validity of the experiment is highly affected.

Ambiguity About Direction of Causal Influence This is the question of whether A causes B, B causes A, or even X causes A and B. An example is if a correlation between program complexity and defect rate is observed. The question is if high program complexity causes high defect rate, or vice versa, or if high complexity of the problem to be solved causes both.

Most of the threats to internal validity can be addressed through the experiment design. For example, by introducing a control group many of the internal threats can be controlled. On the other hand, multiple group threats are introduced instead.

Multiple Groups Threats In a multiple groups experiment, different groups are studied. The threat to such studies is that the control group and the selected experiment groups may be affected differently by the single group threats as defined above. Thus there are interactions with the selection.

Interactions with Selection The interactions with selection are due to different behavior in different groups. For example, the selection–maturation interaction means that different groups mature at different speeds, for example, if two groups apply one new method each. If one group learns its new method faster than the other one does, due to its learning ability, the selected groups mature differently. Selection history means that different groups are affected by history differently, etc.

Social Threats to Internal Validity These threats are applicable to single group and multiple groups experiments. Examples are given below from an inspection experiment where a new method (perspective-based reading) is compared to an old one (checklist-based reading).

Diffusion or Imitation of Treatments This effect occurs when a control group learns about the treatment from the group in the experiment study or they try to imitate the behavior of the group in the study. For example, if a control group uses a checklist-based inspection method and the experiment group uses perspective-based methods, the former group may hear about the perspective-based method and perform their inspections influenced by their own perspective. The latter may be the case if the reviewer is an expert in a certain area.

Compensatory Equalization of Treatments If a control group is given compensation for being a control group, as a substitute for the fact that they do not get treatments, this may affect the outcome of the experiment. If the control group is taught another new method as a compensation for not being taught the perspective-based method, their performance may be affected by that method.

Compensatory Rivalry A subject receiving less desirable treatments may, as the natural underdog, be motivated to reduce or reverse the expected outcome of the experiment. The group using the traditional method may do their very best to show that the old method is competitive.

Resentful Demoralization This is the opposite of the previous threat. A subject receiving less desirable treatments may give up and not perform as well as it generally does. The group using the traditional method is not motivated to do a good job, while learning something new inspires the group using the new method.

9.8.3 Construct Validity

Construct validity concerns generalizing the result of the experiment to the concept or theory behind the experiment. Some threats relate to the design of the experiment, others to social factors.

Design Threats The design threats to construct validity cover issues that are related to the design of the experiment and its ability to reflect the construct to be studied.

Inadequate Pre-operational Explication of Constructs This threat, despite its extensive title, is rather simple. It means that the constructs are not sufficiently defined before they are translated into measures or treatments. The theory is not clear enough, and hence the experiment cannot be sufficiently clear, for example, if two inspection methods are compared and it is not clearly enough stated what being "better" means. Does it mean to find the most faults, the most faults per hour, or the most serious faults?

Mono-Operation Bias If the experiment includes a single independent variable, case, subject, or treatment, the experiment may underrepresent the construct and thus not give the full picture of the theory. For example, if an inspection experiment is conducted with a single document as object, the cause construct is underrepresented.

Mono-Method Bias Using a single type of measure or observation involves a risk that if this measure or observation gives a measurement bias, then the experiment will be misleading. By involving different types of measures and observations they can be cross-checked against each other. For example, if the number of faults found is measured in an inspection experiment where the fault classification is based on subjective judgment, the relations cannot be sufficiently explained. The experimenter may bias the measures.

Confounding Constructs and Levels of Constructs In some relations it is not primarily the presence or absence of a construct, but the level of the construct, which is of importance to the outcome. The effect of the presence of the construct is confounded with the effect of the level of the construct. For example, the presence or absence of prior knowledge in a programming language may not explain the causes in an experiment, but the difference may depend on if the subjects have 1, 3, or 5 years of experience with the current language.

Interaction of Different Treatments If the subject is involved in more than one study, treatments from the different studies may interact. Then you cannot conclude whether the effect is due to either of the treatments or a combination of treatments.

Interaction of Testing and Treatment The testing itself, i.e., the application of treatments, may make the subjects more sensitive or receptive to the treatment. Then the testing is a part of the treatment. For example, if the testing involves measuring

the number of errors made in coding, then the subjects will be more aware of their errors made, and thus try to reduce them.

Restricted Generalizability Across Constructs The treatment may affect the studied construct positively, but unintentionally affect other constructs negatively. This threat makes the result hard to generalize into other potential outcomes. For example, a comparative study concludes that improved productivity is achieved with a new method. On the other hand, it can be observed that it reduces the maintainability, which is an unintended side effect. If the maintainability is not measured or observed, there is a risk that conclusions will be drawn based on the productivity attribute, ignoring the maintainability.

Social Threats to Construct Validity These threats are concerned with issues related to behavior of the subjects and the experimenters. They may, based on the fact that they are part of an experiment, act differently than they do otherwise, which gives false results from the experiment.

Hypothesis Guessing When people take part in an experiment they might try to figure out the purpose and intended result of the experiment. Then they are likely to base their behavior on their guesses about the hypotheses, either positively or negatively, depending on their attitude to the anticipated hypothesis.

Evaluation Apprehension Some people are afraid of being evaluated. A human tendency is to try to look better when being evaluated, which is confounded to the outcome of the experiment. For example, if different estimation models are compared, people may not report their true deviations between estimate and outcome, but some false but "better" values.

Experimenter Expectancy The experimenters can bias the results of a study both consciously and unconsciously based on what they expect from the experiment. The threat can be reduced by involving different people who have no or different expectations of the experiment. For example, questions can be formulated in different ways and the formulation may affect the answer; hence there is a risk that you formulate a question to get the answers you want.

Sjøberg and Rye Bergerson analyzed the state of understanding and reporting practice of construct validity in empirical software engineering [233]. They observed an increasing appearance of construct validity analyses, although they also found a lack of adherence to established definitions. To support construct validity analyses, they defined a reference model, similar to Fig. 9.2. Based on their findings, they propose a set of guidelines to improve understanding and reporting of construct validity:

1. Create a model of constructs under study.
2. Import constructs if possible, from earlier studies.
3. When defining new constructs, (a) ensure adequate concept definitions, (b) avoid overlapping indicators, and (c) avoid bias.

4. Focus on the core constructs.
5. Report threats and constructs explicitly.

9.8.4 External Validity

Threats to external validity are conditions that limit our ability to generalize the results of our experiment to, for example, industrial practice. There are three types of interactions with the treatment: people, place, and time:

Interaction of Selection and Treatment This is an effect of having a subject population not representative of the population we want to generalize to, i.e., the wrong people participate in the experiment. An example of this threat is to select only programmers in an inspection experiment when programmers as well as testers and system engineers generally take part in the inspections.

Interaction of Setting and Treatment This is the effect of not having the experimental setting or material representative of, for example, industrial practice. An example is using old-fashioned tools in an experiment when up-to-date tools are common in industry. Another example is conducting the experiment on toy problems. This means wrong "place" or environment.

Interaction of History and Treatment This is the effect of the experiment being conducted at a special time or on a special day which affects the results. If, for example, a questionnaire is conducted on safety-critical systems a few days after a big software-related crash, people tend to answer differently than a few days before, or some weeks or months later.

The threats to external validity are reduced by making the experimental environment as realistic as possible. On the other hand, reality is not homogeneous. Most important is to characterize and report the characteristics of the environment, such as staff experience, tools, and methods to evaluate the applicability in a specific context.

9.9 Priority Among Types of Validity Threats

There is a conflict between some of the types of validity threats. The four types considered are internal validity, external validity, conclusion validity, and construct validity. When increasing one type, another type may decrease. Prioritizing among the validity types is hence an optimization problem, given a certain purpose of the experiment.

For example, using undergraduate students in an inspection experiment will probably enable larger study groups, reduce heterogeneity within the group, and give reliable treatment implementation. This results in high conclusion validity,

while the external validity is reduced, since the selection is not representative if we want to generalize the results to the software industry.

Another example is to have the subjects measure several factors by filling out schemes to make sure that the treatments and outcomes really represent the constructs under study. This action will increase construct validity, but there is a risk that conclusion validity will be reduced since more tedious measurements have a tendency to reduce the reliability of the measures.

In different experiments, different types of validity can be prioritized differently, depending on the purpose of the experiment. Siegmund et al. [229] launched an online survey to which they invited program committee and editorial board members of 11 major software engineering venues. Out of the 807 invited participants, 94 completed the questionnaire (10% response rate). For human-oriented studies, a majority (47%) preferred external validity over internal validity, while a minority (15%) preferred the opposite. For technology-oriented studies, 32% prioritized external validity while 16% prioritized internal validity.

Verdecchia et al. [259] outlined a vision for threats to validity (TTV) analyses in software engineering, "considering TTV in every phase[1] of the study, including identifying potential threats, designing mitigation strategies, and documenting TTV appropriately." This implies that threats to validity are not a reflection after the fact, but part of a proactive and deliberate study design process, in relation to the stated goals for this specific study.

Cook and Campbell [50] propose the following priorities for studies with the goal of theory testing and applied research, respectively:

- In theory testing, it is most important to show that there is a casual relationship (internal validity) and that the variables in the experiment represent the constructs of the theory (construct validity). Adding to the experiment size can generally solve the issues of statistical significance (conclusion validity). Theories are seldom related to specific settings, population, or times to which the results should be generalized. Hence there is limited need for external validity concerns. The priorities for experiments in theory testing are in decreasing order: internal, construct, conclusion, and external.
- In applied research, which is the target area for most of the software engineering experiments, the priorities are different. Again, the relationships under study are of highest priority (internal validity) since the key goal of the experiment is to study relationships between causes and effects. In applied research, the generalization—from the context in which the experiment is conducted to a wider context—is of high priority (external validity). For a researcher, it is not so interesting to show a particular result for company X, but rather that the result is valid for companies of a particular size or application domain. Furthermore, the applied researcher is relatively less interested in which of the components in a complex treatment really causes the effect (construct validity). For example,

[1] Here, we refer to it as "step," for example, every step in the experiment process.

in a reading experiment, it is not so interesting to know if it is the increased understanding in general by the reviewer or if it is the specific reading procedure that helps the readers to find more faults. The main interest is in the effect itself. Finally, in practical settings it is hard to get sufficient size of data sets; hence the statistical conclusions may be drawn with less significance (conclusion validity). The priorities for experiments in applied research are in decreasing order: internal, external, construct, and conclusions.

It can be concluded that the threats to validity of experimental results are important to evaluate and balance during planning of an experiment. Depending on the purpose of the experiment, different validity types are given different priority. The threats to an experiment are also closely related to the practical importance of the results. We may, for example, be able to show a statistical significance, but the difference is of no practical importance. This issue is further elaborated in Sect. 11.3.15.

9.10 Example Experiment

This description is a continuation of the example introduced in Sect. 8.2. The input to the planning step is the goal definition. Some of the issues related to planning have partially been addressed in the way the goal definition is formulated in the example. It is already stated that students will be the subjects and the text also indicates that the experiment will involve more than one requirements document. Planning is a key step when conducting an experiment. A mistake in the planning step may affect the whole outcome of the experiment. The planning step includes seven sub-steps as shown in Fig. 9.1.

Context Selection The type of context is in many cases at least partially decided by the way the goal definition is formulated. It is implicitly stated that the experiment will be run offline, although it could potentially be part of a student project, which would have meant online, or in real life for the students. However, it would not be conducted as part of an industrial development project. It is essential to observe that real life refers to the real life of the subjects and where we can assume that the findings are relevant. The experiment will be run with a mixture of MSc and PhD students.

An offline experiment with students implies that it may be difficult to have time to inspect a requirements document for a fully-fledged real system. In many cases, experiments of this type have to resort to a requirements document with limited features. In this specific case, two requirements documents from a lab package will be used. The choice to use two requirements documents has some implications when it comes to the choice of design type, which we will come back to. The requirements documents have some limitations when it comes to features and hence they are to some extent to be considered as “toy” requirements documents.

The experiment can be considered as general in the sense that the objective is to compare two reading techniques in general (from a research perspective), and it is not about comparing an existing reading technique in a company with a new alternative reading technique. The latter would have made the experiment specific for the situation at the company. In both these cases, there are some issues to take into account to ensure a fair comparison.

In the general research case, it is important that the comparison is fair in the sense that the support for the two techniques being investigated is comparable. It is of course easy to find a very poor checklist and then provide good support for Perspective-Based Reading (PBR). This would favor PBR and hence the outcome of the experiment would definitively be challenged. This is also the reason why having "no support" is not a good control. An experimental comparison/evaluation must be based on having two comparable methods with similar support. Using "no support" as a control group should be avoided. It would only be interesting if the group having support performs worse than those not having any support, or it is the "old" way of working at a company. However, this situation is quite rare and hence it is rarely worth performing an experiment under these circumstances.

In the specific case, there is no problem with fairness in the type of support provided, since as long as an existing technique is compared with a new alternative, then it is fine from a support perspective. The main challenge in the specific case is that the participants know the existing technique very well, while a new technique must be taught to them. Thus, the new technique may have a disadvantage since it is not as well known. On the other hand, it has the advantage of potentially being more interesting to the subjects, since it means learning a new technique. Thus, in this case the situation is not that clear-cut, but the potential biases in favor of one or the other technique must be taken into consideration by the researcher.

Hypothesis Formulation In the goal definition it is expressed that we would like to compare both effectiveness and efficiency when it comes to detecting faults when using two different reading techniques when conducting the inspection. The first method is Perspective-Based Reading (PBR) and the second method is Checklist-Based Reading (CBR). PBR is based on the reviewers having different perspectives when performing the inspection. CBR is based on having a checklist for different items that are likely to relate to faults in requirements documents.

The fact that the requirements documents to be used in the experiment have been used in prior experiments means that the number of faults is assumed to be known, although it cannot be ruled out that new faults will be found. As mentioned in the scoping of the experiment, effectiveness refers to the number of faults found out of the total number of faults, while efficiency also includes time, i.e., whether more faults are found per time unit. To be able to formulate the formal hypotheses, we let N be the number of faults and Nt the number of faults found per time unit.

If we let:

- μ_{NPBR} and μ_{NCBR} be the number of faults found using PBR and CBR respectively, and

- μ_{NtPBR} and μ_{NtCBR} be the number of faults found per time unit using PBR and CBR respectively.

Then, the hypotheses are formulated as follows:

Effectiveness:
$H_0 : \mu_{NPBR} = \mu_{NCBR}$
$H_1 : \mu_{NPBR} <> \mu_{NCBR}$

It should be noted that we have chosen the alternative hypothesis as being any difference between the two reading techniques. In other words, the alternative hypothesis is formulated as a two-sided hypothesis with no assumption regarding one technique being better than the other.

Efficiency:
$H_0 : \mu_{NtPBR} = \mu_{NtCBR}$
$H_1 : \mu_{NtPBR} <> \mu_{NtCBR}$

The hypotheses mean that we would like to show with a statistical significance that the two reading techniques find a different number of faults and a different number of faults are found per time unit. We would like to refute the null hypothesis. It must be noted that not being able to refute the null hypothesis does *not* imply accepting the null hypothesis. This type of outcome may be due to having too few subjects and not due to the reading techniques being equally good at detecting faults.

Variables Selection The independent variable is the reading technique and it has two levels: PBR and CBR. The dependent variables are the number of faults found and the number of faults found per time unit. This means that we must ensure that the subjects can clearly mark faults found so that the researcher can compare the faults marked with the known set of faults. Furthermore, we must ensure that the subjects can keep track of time and fill in the time when a specific fault was found. It must be noted that it is important to keep track of the time for a specific fault, since a fault may be a false positive and hence we must know which time should be removed from the data set too.

Selection of Subjects It would be preferable to find subjects for the experiment at random. However, in most experiments the researcher tends to be forced to use subjects that are available. This means often students participating in courses at the university become the subjects in experiments run at the university, which is the case in this example experiment. In this case, it is important that the subjects still have the freedom to refuse participation, without any penalty for the individual. If the participation in the experiment gives course credit points, alternative options should be provided.

If the purpose of the experiment is to compare how the two student groups perform using the different methods, then the treatment in the experiment is ruled by the selection of subjects, i.e., the characteristics of the student groups. In fact, this would make it a quasi-experiment. Independently, it is important to characterize the selected subjects to help in assessing the external validity of the study.

Choice of Design Type Once we know which subjects are going to participate, it is time to move on to randomization and decide how the subjects should be divided into groups. A good approach is often to use a pre-test to try to capture the experience of the subjects and based on the outcome of the pre-test divide the subjects into experience groups from which we randomly select subjects to the groups in the experiment. This is done to try to ensure that the groups are as equal as possible when it comes to previous experiences, still maintaining the randomization over the subjects. This is referred to as blocking, i.e., we block on previous experience to try to ensure that it does not affect the outcome of the experiment. Finally, the objective is in most cases to have equally large groups, i.e., we want a balanced design. The choice of design type may be affected by the number of subjects available. If we have many subjects, it is possible to consider more experimental combinations or consider using each subject for only one treatment. With relatively few subjects, it becomes more challenging to design the experiment and to use the subjects wisely without compromising the objectives of the experiment.

Next the design type should be selected. The experiment includes one factor of primary interest (reading technique) with two treatments (PBR and CBR), and a second factor that is not really of interest in the experiment (the requirements documents). Based on the previous decisions taken, the natural design is a completely randomized design where each group first uses either PBR or CBR on one of the requirements documents and then uses the other reading technique on the other requirements document. However, decisions have to be taken on order too. We have two options: (1) either have both groups using different reading techniques on one of the requirements documents first and then switch reading techniques when inspecting the other requirements document, or (2) have both groups using the same reading technique on different requirements documents. In either case, there is an ordering issue. In the first case, one of the requirements documents will be used before the other and in the second case one reading technique will be used before the other. Thus, we have to consider which poses the fewest threats to the experiment. Validity threats are further elaborated below.

Another design option would have been to allow one group to use PBR on a requirements document and the other group to use CBR on the same document. The advantage would be that a larger requirements document could be used in the same time frame. The downside is that only half as many data points are generated. In an experiment it is often the case that a certain amount of time is available for running the experiment. Thus, it becomes a question of how to use the time in the most effective way, i.e., to get as good output from the experiment as possible to address the hypotheses stated. The choice of design is very important and it is always a trade-off. Different types of designs have different advantages and disadvantages. Furthermore, the choice also forms the basis for which statistical method can be applied on the data. This is further discussed in Sect. 11.4.

In this specific case, a completely randomized design is chosen. One group is first assigned to using PBR on the first requirements document and the other group

is assigned to use CBR on the same requirements document. This alternative is chosen since it is believed that an order between the reading techniques is worse than an order between the requirements documents. This is particularly the case since the primary interest is in the difference between reading techniques and not any differences between the two requirements documents.

Instrumentation Given that the experiment is based on a lab package, the requirements documents are already available and hence also a list of detected faults (at least known so far). Otherwise, suitable requirements documents should be identified, preferably with a known number of faults to be able to determine the effectiveness of the reading technique.

The guidelines for the two reading techniques must be developed or reused from elsewhere. Here it is important to ensure a fair comparison, as mentioned above, by providing comparable support for the two methods.

Forms for filling out faults found must be developed or reused from another experiment. It is crucial to ensure traceability between the requirements document and the form, for example by numbering the faults in the requirements document while capturing the information about the fault in the form.

Validity Evaluation Finally, the validity threats must be evaluated. This is important to do upfront to ensure that the threats are minimized. It is close to impossible to avoid all threats. Having said that, it still means that if possible all threats should be identified and whenever possible mitigated.

The evaluation of the threats in this specific example is left as an exercise (see exercise 9.4 in Sect. 9.11).

Next Step in the Experiment Process Based on the steps described above for the example, it is hoped that we are ready to run the experiment. However, before doing so it is recommended that some colleagues review the experiment design. Furthermore, it is good if it is possible to run a trial run of the experiment, although it means using one or more persons who otherwise could have been subjects in the experiment. Thus, it is important to use potential subjects wisely.

9.11 Exercises

9.1 What are a null hypothesis and an alternative hypothesis, and why can a null hypothesis never be accepted?

9.2 What are type-I-error and type-II-error respectively, and which is worse and why?

9.3 What different types of experiment designs are available, and how does the design relate to the statistical methods to apply in the analysis?

9.4 Which are the threats (consider all four types of validity threats) that exist in the example in Sect. 9.10 and explain why they are threats, and what is the trade-off between the different validity types?

9.5 Suggest a design for the experiment used to illustrate the decision-making structure in Sect. 2.5 using what you have learned in this chapter.

Chapter 10
Operation

When an experiment has been designed and planned it must be carried out to collect the data that should be analyzed. This is what we mean by the operation of an experiment. In the operation step of an experiment, the treatments are applied to the subjects. This means that this part of the experiment is where the experimenter actually meets the subjects. In most experiments in software engineering there are only a few other times when the subjects actually are involved. These occasions can, for example, be in a briefing before subjects commit to participate in the experiment and after the experiment when the results of the experiment are presented to the subjects. In most cases, experiments in software engineering deal with humans, although it is possible to run technology-oriented experiments too, as discussed in Sect. 6. This chapter deals to some extent with how to motivate people to participate and take part in experiments.

Even if an experiment has been perfectly designed and the collected data is analyzed with the appropriate analysis methods, the result will be invalid if the subjects have not participated seriously in the experiment. Since the field of experimental psychology also deals with experiments involving humans, guidelines for conducting experiments from that field [2, 41] are to some extent also applicable in software engineering.

The operation step of an experiment consists of three sub-steps: *preparation* where subjects are chosen and forms, etc. are prepared, *execution* where the subjects perform their tasks according to different treatments and data is collected, and *data validation* where the collected data is validated. The three sub-steps are displayed in Fig. 10.1 and they are further described in this chapter.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_10

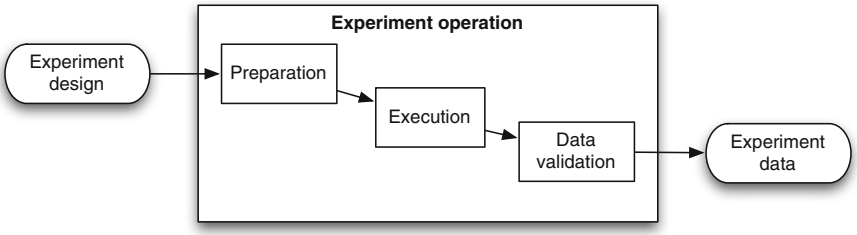


Fig. 10.1 Three sub-steps in experiment operation

10.1 Preparation

Before the experiment is executed there are some preparations that have to be made. The better these preparations are performed, the easier it will be to execute the experiment. There are two important aspects in the preparation. The first is to select and inform participants, and the second is to prepare material such as forms and tools.

10.1.1 Commit Participants

Before an experiment can be started, people who are willing to act as subjects have to be found. It is essential that the people are motivated and willing to participate throughout the whole experiment.

In many cases it is important to find people who work with tasks in the experiment that are similar to their ordinary work tasks. For example, if an experiment involves writing C-code with different kinds of tools, it would probably make sense to involve persons who are used to writing C-code, and not to involve Java programmers. If people are chosen who are not a representative set of the people who we want to be able to make statements about, this will be a threat to the external validity of the experiment (see Sect. 9.7). The selection of subjects, in terms of sampling technique, is discussed in Sect. 9.4.

When the right people are found, it is necessary to convince these people to participate in the experiment. Several ethical aspects have to be considered when people are participating as subjects.

Obtain Consent The participants have to agree to the research objectives. If the participants do not know the intention of the work or the work does not comply with what they thought they should do when they agreed to participate, there is a risk that they will not perform the experiment according to the objectives and their personal ability. This could result in the data becoming invalid. It is important to describe how the result of the experiment will be used and published. It should be

made clear to the participants that they are free to withdraw from the experiment. Sometimes a trade-off must be made between this aspect and the design with respect to validity. If the participants are affected by the experiment as such, this will affect the validity of the experiment.

Sensitive Results If the results obtained in the experiment are sensitive for the participants, it is important to assure the participants that the results of their personal performance in the experiment will be kept confidential. It is sometimes hard to judge if the result is sensitive or not, but generally it can be said that if the result would have a meaning for the participants outside the experiment it is in some way sensitive. For example, if the experiment measures the productivity of a programmer, the result would indicate how skilled the programmer is, and hence the result would be sensitive. On the other hand, if participants are asked to use a method for acceptance testing and they normally never deal with this type of testing, the result of the experiment would probably not be sensitive.

Inducements One way to attract people to an experiment is to offer some kind of inducement. The value of this inducement should, however, not be too high, since this could cause people to participate merely to receive the inducement. This would not motivate people to participate seriously in the experiment.

Disclosure Disclosure means to reveal all details of the experiment as openly as possible to the experiment subjects. The opposite, to deceive or betray the participants, is generally not acceptable. If alternative ways of conducting the experiment are available these methods should be used instead. If non-disclosure is the only alternative, it should only be applied if it concerns aspects that are insignificant to the participants and do not affect their willingness to participate in the experiment. In case of partial disclosure, the situation should be explained and revealed to the participants as early as possible.

For more discussion on ethical aspects in experimentation, see Sect. 3.1.

10.1.2 Instrumentation Concerns

Before the experiment can be executed, all experiment instruments must be ready (see Sect. 9.6). This may include the experiment objects, guidelines for the experiment, and measurement forms and tools. The required instruments are determined by the design of the experiment and the method that will be used for data collection.

If the subjects themselves need to collect data, this means in most cases that some kind of form must be handed out to the participants. One thing to determine when forms are constructed is whether they should be personal or the participants should fill them out anonymously. If there will be no additional studies and hence there is no real need for the experimenter to distinguish between different participants, it may be appropriate to use anonymous forms. This will, however, mean that there is no possibility to contact the participant if something is filled out in an unclear way.

In many cases it is appropriate to prepare one personal set of instruments for every participant. This is because many designs deal with randomization and repeated tests, such that different participants should be subject to different treatments. This can be done also when the participants are anonymous.

If data needs to be collected in interviews, questions should be prepared before the execution of the experiment. Here it may also be appropriate to prepare different questions for different participants.

10.2 Execution

The experiment can be executed in a number of different ways. Some experiments, such as simple inspection experiments, can be carried out on one occasion when all participants are gathered at, for example, a meeting. The advantage of this is that the result of the data collection can be obtained directly at the meeting and there is no need to contact the participants and later on ask for their respective results. Another advantage is that the experimenter is present during the meeting and if questions arise they can be resolved directly.

Some experiments are, however, executed during a much longer time span, and it is impossible for the experimenter to participate in every detail of the experiment and the data collection. This is, for example, the case when the experiment is performed in relation to one or several large projects, where different methods for development are evaluated. An example of such an experiment is presented by Ohlsson and Wohlin [187], where a course in large-scale software development was studied during two years. Each year, seven projects were run in parallel with a total of approximately 120 students. The objective of the experiment by Ohlsson and Wohlin [187] was to evaluate different levels of formality when collecting effort data.

10.2.1 Data Collection

Data can be collected manually by the participants that fill out forms, manually supported by tools, in interviews, or automatically by tools.

An advantage of using forms is that it does not require so much effort from the experimenter, since the experimenter does not have to actively take part in the collection. A drawback is that there is no possibility for the experimenter to directly reveal inconsistencies, uncertainties, and flaws in the forms, etc. This type of faults cannot be revealed until after the data collection or if the participants bring attention to faults or have questions. An advantage with interviews is that the experimenter has the possibility to communicate better with the participants during the data collection. A drawback is of course that it requires more effort from the experimenter.

10.2.2 Experimental Environment

If an experiment is performed within a regular development project, the experiment should not affect the project more than necessary. This is because the reason for performing the experiment within the project is to see the effects of different treatments in an environment such as the one in the project. If the project environment is changed too much because of the experiment that effect will be lost.

There are, however, some cases where it is appropriate to have some interaction between the experiment and the project. If the experimenter, for example, reveals that some parts of the project could be performed better or that estimations are not correct, it would be appropriate for the experimenter to tell the project leader. This type of direct feedback from the experiment to the project can help to motivate project personnel to participate in the experiment.

10.3 Data Validation

When data has been collected, the experimenter must check that the data is reasonable and that it has been collected correctly. This deals with aspects such as if the participants have understood the forms and therefore filled them out correctly. Another source of error is that some participants may not have participated in the experiment seriously and some data therefore should be removed before the analysis. Outlier analysis is further discussed in Sect. 11.2.

It is important to review whether the experiment has been conducted in the way that was intended. It is, for example, essential that the subjects have applied the correct treatments in the correct order. If this type of misunderstanding has occurred, the data is of course invalid.

One way to check that the participants have not misunderstood the intentions of the experimenter is to give a seminar, or in some other way present the results of the data collection. This will give the participants the possibility to reflect on results that they do not agree with. It also helps in building long-term trust, as discussed in Sect. 3.1

10.4 Example Operation

The experiment design from Sect. 9.10 is the input to the operation, which consists of three sub-steps that must be addressed.

Preparation First of all the subjects must be identified. In this example, PhD and MSc students are invited as subjects. Once there is a potential set of participants, it is important to convince them to participate and get their commitment to participate in the experiment. After having an initial commitment, consent must be ensured from

the participants. It is recommended that consent forms be used even if the formal rules may not require it. Other issues to take into account in relation to ethics are described in Sect. 10.1.1. Assigning subjects to treatments must be done using a randomization procedure. If the design includes a blocking factor (type of student), subjects should be split according to that factor, and then randomly assigned to treatments within each blocking group. If a balanced design is chosen, the selection must end up in the same number of subjects for each group.

Next, it is essential to ensure that the infrastructure needed is in place. This includes having a suitable room booked, providing sufficient distance between the subjects. Copies of all documents and forms must be available for all subjects. Given that time is going to be collected, a clock is needed in the room. It cannot be assumed that everybody has access to his or her own clock.

Execution During the execution it is essential to ensure the participants are suitably spread out in the room. As it is an inspection experiment, it should be possible to run the experiment once with all subjects doing the inspection at the same time. This also means that it is easy to provide support for any questions that may arise during the experiment. Depending on whether the data will be collected by filling in forms by hand or by use of a computer, preparation has to be done accordingly.

Data Validation Finally, the data has to be validated. It may be the case that one or several subjects leave the experiment very early and their data forms have to be checked carefully to ensure that they have filled in the forms in a reasonable way. Furthermore, it must be checked that everybody has understood how to fill in the data in a correct way. If this is not the case, data from one or several subjects might need to be removed.

10.5 Exercises

10.1 Which factors should be considered when selecting subjects?

10.2 Why are ethical issues important in experimentation?

10.3 Why is it necessary to prepare the instrumentation carefully before an experiment?

10.4 What is data validation and why should it be done before the statistical analysis?

10.5 How should we handle subjects that have a personal interest in the outcome of the experiment?

Chapter 11
Analysis and Interpretation

The experiment data from the operation is input to the analysis and interpretation. After collecting experimental data in the operation step, we want to be able to draw conclusions based on this data. To be able to draw valid conclusions, we must interpret the experiment data. Quantitative interpretation may be carried out in three sub-steps, as depicted in Fig. 11.1.

In the first sub-step, the data is characterized using *descriptive statistics*, which visualize central tendency, dispersion, etc. In sub-step two, abnormal or false data points are excluded, thus *reducing the data set* to a set of valid data points. In the third sub-step, the data is analyzed by *hypothesis testing*, where the hypotheses of the experiment are evaluated statistically, at a given level of significance. These sub-steps are described in more detail in the remainder of this chapter.

11.1 Descriptive Statistics

Descriptive statistics deal with the presentation and numerical processing of a data set. After collecting experimental data, descriptive statistics may be used to describe and graphically present interesting aspects of the data set. Such aspects include measures indicating, for example, where on some scale the data is positioned and how concentrated or spread out the data set is. The goal of descriptive statistics is to get a perception of how the data set is distributed. Descriptive statistics may be used before carrying out hypothesis testing, in order to better understand the nature of the data and to identify abnormal or false data points (so-called outliers).

In this section, we present a number of descriptive statistics and plotting techniques that may help to get a general view of a data set. The scale of measurement (see Sect. 3.4) restricts the type of statistics that are meaningful to compute. Table 11.1 shows a summary of some of these statistics in relation to the scales under which they are admissible. It should, however, be noted that measures

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_11

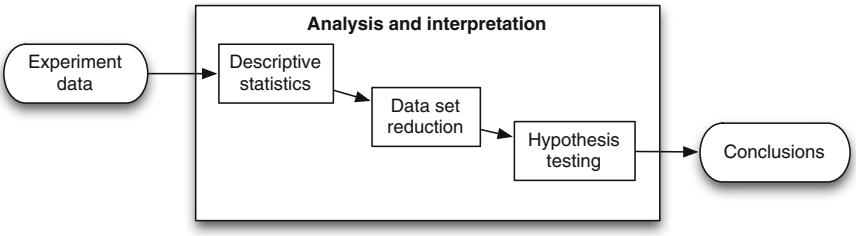


Fig. 11.1 Three sub-steps in quantitative interpretation

Table 11.1 Some relevant statistics for each scale

	Measure of		
Scale type	**central tendency**	**dispersion**	**dependency**
Nominal	mode	frequency	
Ordinal	median, percentile	interval of variation	Spearman corr. coeff., Kendall corr. coeff.
Interval	mean, variance, and range	standard deviation	Pearson corr. coeff.
Ratio	geometric mean	coefficient of variation	

of one scale type can be applied to the more powerful scales, for example, mode can be used for all four scales in Table 11.1.

11.1.1 Measures of Central Tendency

Measures of central tendency, such as mean, median, and mode, indicate a "middle" of a data set. This "midpoint" is often called average and may be interpreted as an estimation of the *expectation* of the stochastic variable from which the data points in the data set are sampled.

When describing the measures of central tendency, we assume that we have n data points $x_1 \ldots x_n$, sampled from some stochastic variable. The (arithmetic) *mean*, denoted $\bar{x}$, is calculated as:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

The mean value is meaningful for the interval and ratio scales. For example, we may compute the mean for the data set $(1,\ 1,\ 2,\ 4)$ resulting in $\bar{x} = 2.0$.

The *median*, denoted $\tilde{x}$, represents the middle value of a data set, following that the number of samples that are higher than the median is the same as the number of samples that are lower than the median. The median is calculated by sorting the

samples in ascending (or descending) order and picking the middle sample. This is well defined if n is odd. If n is even, the median may be defined as the arithmetic mean of the two middle values. The latter operation requires that the scale is at least interval. If the scale is ordinal, one of the two middle values may be selected by random choice, or the median may be represented as a pair of values.

The median value is meaningful for the ordinal, interval, and ratio scales. As an example, we may compute the median for the data set (1, 1, 2, 4) resulting in $\tilde{x} = 1.5$.

The median is a special case of the *percentile*, namely the 50% percentile, denoted $x_{50\%}$, indicating that 50% of the samples lie below $x_{50\%}$. In general x_p denotes the percentile where $p\%$ of the samples lie below this value. The percentile value is meaningful for the ordinal, interval, and ratio scales.

The *mode* represents the most commonly occurring sample. The mode is calculated by counting the number of samples for each unique value and selecting the value with the highest count. The mode is well defined if there is only one value that is more common than all the others. If an odd number of samples have the same occurrence count, the mode may be selected as the middle value of the most common samples. The latter operation requires that the scale is at least ordinal. If the scale is nominal, the mode may be selected among the most common samples by random choice or represented as a pair of the most common values.

The mode value is meaningful for the nominal, ordinal, interval, and ratio scales. As an example, we may compute the mode for the data set (1, 1, 2, 4) giving a mode of 1.

A less common measure of central tendency is the *geometric mean*, which is calculated as the n:th root of the product of all samples, as shown below.

$$\sqrt[n]{\prod_{i=1}^{n} x_i}$$

The geometric mean is well defined if all samples are non-negative and meaningful for the ratio scale. The (arithmetic) mean and median are equal if the distribution of samples is symmetric. If the distribution is both symmetric and has one unique maximum, all these three measures of central tendency are equal. However, if the distribution of samples is skewed, the values of the mean, median, and mode may differ (see Fig. 11.2).

If, for example, the higher tail of the distribution is long, the mean is increased, while the median and mode are unaffected. This indicates that the mean is a more sensitive measure. However, it requires at least an interval scale, and hence may not always be meaningful.

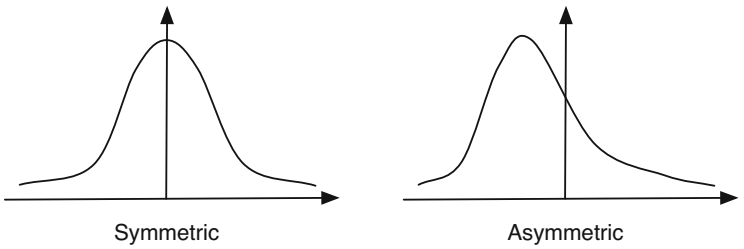


Fig. 11.2 A symmetric distribution has the same values of mean, median, and mode, while they may differ if the distribution is asymmetric

11.1.2 Measures of Dispersion

The measures of central tendency do not convey information on the dispersion of the data set. Thus, it is necessary to measure the level of variation from the central tendency, i.e., to see how spread or concentrated the data is. The (sample) *variance*, denoted s^2, is a common measure of dispersion, and is calculated as follows:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Hence, the variance is the mean of the square distance from the sample mean. It may seem odd that the dividend is $n-1$ and not just n, but by dividing with $n-1$, the variance gets some desirable properties. In particular, the sample variance is an unbiased and consistent estimation of the variance of the stochastic variable. The variance is meaningful for the interval and ratio scales.

The *standard deviation*, denoted s, is defined as the square root of the variance:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

The standard deviation is often preferred over the variance as it has the same dimension (unit of measure) as the data values themselves. The standard deviation is meaningful for the interval and ratio scales.

The *range* of a data set is the distance between the maximum and minimum data values:

$$\text{range} = x_{max} - x_{min}$$

The range value is meaningful for the interval, and ratio scales.

Table 11.2 A frequency table example

Value	Frequency	Relative frequency
1	3	23%
2	2	15%
3	1	8%
4	3	23%
5	1	8%
6	2	15%
7	1	8%

The *variation interval* is represented by the pair $(x_{min},\ x_{max})$ including the minimum and maximum of the data values. This measure is meaningful for ordinal, interval and ratio scales.

Sometimes the dispersion is expressed in percentages of the mean. This value is called the *coefficient of variation* and is calculated as follows:

$$100\frac{s}{x}$$

The coefficient of variation measure has no dimension and is meaningful for the ratio scale.

A general view of dispersion is given by the *frequency* of each data value. A frequency table is constructed by tabulating each unique value and the count of occurrence for each value. The *relative frequency* is calculated by dividing each frequency by the total number of samples. For the data set (1, 1, 1, 2, 2, 3, 4, 4, 4, 5, 6, 6, 7) with 13 samples we can construct the frequency table shown in Table 11.2. The frequency is meaningful for all scales.

11.1.3 Measures of Dependency

When the data set consists of related samples in pairs (x_i, y_i) from two stochastic variables, X and Y, it is often interesting to examine the dependency between these variables.

If X and Y are related through some function, $y = f(x)$, we want to estimate this function. If we suspect that the function $y = f(x)$ is linear and could be written on the form $y = \alpha + \beta x$, we could apply *linear regression*. Regression means fitting the data points to a curve, and in our case we will show how fitting a line that minimizes the sum of the quadratic distances to each data point makes linear

regression. Before we present the formulas we define the following shorthand for some commonly occurring sums:

$$S_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$S_{yy} = \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$S_{xy} = \sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) = (\sum_{i=1}^{n} x_i y_i) - \frac{1}{n}(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)$$

The sums can be used to compute the regression line $y = \bar{y} + \beta(x - \bar{x})$ where the slope of the line is

$$\beta = \frac{S_{xy}}{S_{xx}}$$

and the line crosses the y-axis at $\alpha = \bar{y} - \beta\bar{x}$.

If the dependency is non-linear, it may be possible to find a *transformation* of data, so that the relation becomes linear, and linear regression can be used. If, for example, the relation is exponential, $y = \alpha x^{\beta}$, this implies that a logarithmic transformation of the data results in the linear relation $log(y) = log(\alpha) + \beta log(x)$. After the logarithmic transformation we can use linear regression to calculate the parameters of the line.

For a single number that quantifies how much two data sets, x_i and y_i, vary together, we can use the *covariance*. This measure of dependency, denoted c_{xy}, is defined as follows:

$$c_{xy} = \frac{S_{xy}}{n-1}$$

The covariance is meaningful for interval and ratio scales. The covariance is dependent on the variance of each variable, and to be able to compare dependencies between different related variables, the covariance may be normalized with the standard deviations of x_i and y_i. If we do this we get the *correlation coefficient* r (also called *Pearson correlation coefficient*), which is calculated as follows:

$$r = \frac{c_{xy}}{s_x s_y} = \frac{S_{xy}}{\sqrt{S_{xx} S_{yy}}} = \frac{(n\sum_{i=1}^{n} x_i y_i) - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{\sqrt{(n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2)(n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2)}}$$

The r-value is between -1 and $+1$, and if there is no correlation r equals zero. The opposite is, however, not true. The x_i and y_i values may be strongly correlated

in a non-linear manner even if $r = 0$. The (Pearson) correlation coefficient measures only linear dependency and is meaningful if the scales of x_i and y_i are interval or ratio, and works well for data that is normally distributed.

If the scale is ordinal or if the data is far from normally distributed, the *Spearman rank-order correlation coefficient*, denoted r_s, can be used. The Spearman correlation is calculated in the same manner as the Pearson correlation except that the ranks (i.e., the order numbers when the samples are sorted) are used instead of the sample values (see, e.g., Siegel and Castellan [228]).

Another measure of dependency is the *Kendall rank-order correlation coefficient*, denoted T. The Kendall correlation is suitable as a measure for the same sort of data as the Spearman correlation, i.e., at least ordinal samples in a pair. The Kendall correlation differs, however, in the underlying theory as it focuses on counting agreements and disagreements in ranks between samples (see, e.g., Siegel and Castellan [228]).

If we have more than two variables, we can apply *multivariate analysis*, including techniques such as *multiple regression, principal component analysis (PCA), cluster analysis*, and *discriminant analysis*. These techniques are described by, for example, Manly [165] and Kachigan [124, 125].

11.1.4 Graphical Visualization

When describing a data set, quantitative measures of central tendency, dispersion, and dependency should be combined with graphical visualization techniques. Graphs are very illustrative and give a good overview of the data set.

One simple but effective graph is the *scatter plot*, where pairwise samples (x_i, y_i) are plotted in two dimensions, as shown in Fig. 11.3.

The scatter plot is good for assessing dependencies between variables. By examining the scatter plot, it can be seen how spread or concentrated the data points are, and if there is a tendency toward linear relation. Atypical values (outliers) may be identified, and the correlation may be observed. In Fig. 11.3, there is a linear tendency with a positive correlation, and we may observe potential outliers. In this particular case there is one candidate outlier.

The *box plot* is good for visualizing the dispersion and skewedness of samples. The box plot is constructed by indicating different percentiles graphically, as shown

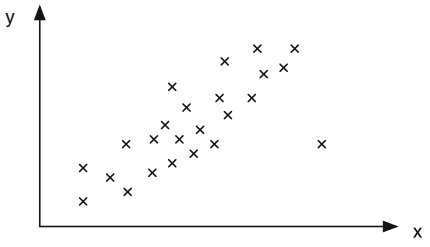


Fig. 11.3 A scatter plot

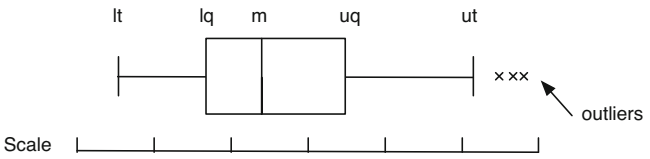


Fig. 11.4 A box plot

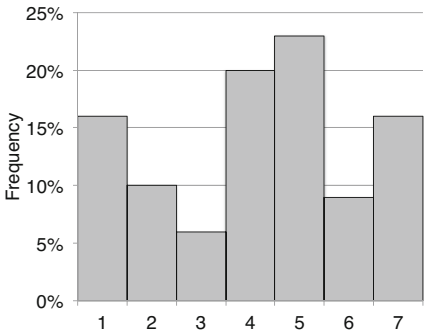


Fig. 11.5 A histogram

in Fig. 11.4. Box plots can be made in different ways. We have chosen an approach advocated by, for example, Fenton and Pfleeger [76] and Frigge et al. [81]. The main difference between the approaches is how to handle the whiskers. Some literature proposes that the whiskers should go to the lowest and highest values respectively (see, e.g., Montgomery [180]. Fenton and Pfleeger [76] propose using a value which is the length of the box, multiplied by 1.5 and added to or subtracted from the upper and lower quartiles, respectively.

The middle bar in the box m is the median. The lower quartile lq is the 25% percentile (the median of the values that are less than m), and the upper quartile uq is the 75% percentile (the median of the values that are greater than m). The length of the box is $d = uq - lq$.

The tails of the box represent the theoretical bound within which it is likely to find all data points if the distribution is normal. The upper tail ut is $uq + 1.5d$ and the lower tail lt is $lq - 1.5d$ [81]. The tail values are truncated to the nearest actual data point, in order to avoid meaningless values (such as negative lines of code).

Values outside the lower and upper tails are called *outliers*, and are shown explicitly in the box plot. In Fig. 11.4, there are three outliers.

The *histogram* can be used to give an overview of the distribution density of the samples from one variable. A histogram consists of bars with heights that represent the frequency (or the relative frequency) of a value or an interval of values, as shown in Fig. 11.5. The histogram is thus a graphical representation of a frequency table. One distribution of particular interest is the normal distribution, since it is one aspect that should be taken into account when analyzing the data. Thus, a plot could

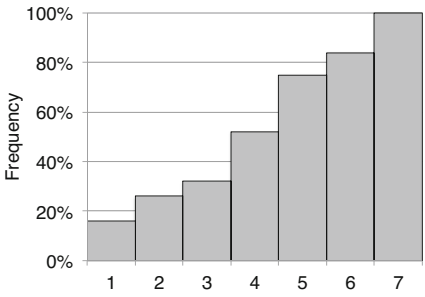


Fig. 11.6 A cumulative histogram

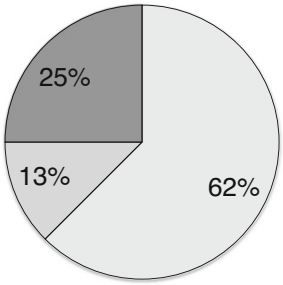


Fig. 11.7 A pie chart

provide a first indication as to whether the data resembles a normal distribution or not. It is also possible to test the data for normality. This is further discussed in Sect. 11.3 when introducing the Chi-square test.

The *cumulative histogram*, illustrated in Fig. 11.6, may be used to give a picture of the probability distribution function of the samples from one variable. Each bar is the cumulative sum of frequencies up to the current class of values.

A *pie chart*, as illustrated in Fig. 11.7, shows the relative frequency of the data values divided into a specific number of distinct classes, by constructing segments in a circle with angles proportional to the relative frequency.

11.2 Data Set Reduction

In Sect. 11.3, a number of statistical methods are described. All methods have in common that the result of using them greatly depends on the quality of the input data. If the data to which the statistical methods are applied does not represent what we think it represents, then the conclusions that we draw from the results of the methods are of course not correct.

Errors in the data set can occur either as systematic errors or as outliers, which means that the data point is much larger or much smaller than one could expect looking at the other data points (see Fig. 11.8).

One effective way to identify outliers is to draw scatter plots, as shown in Fig. 11.8. Another way is to draw box plots, as illustrated in Fig. 11.4. There are

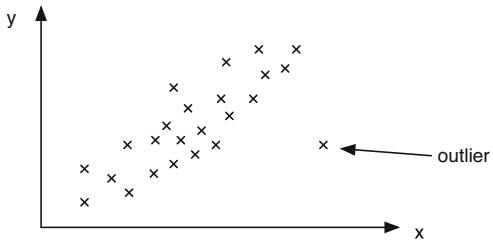


Fig. 11.8 An outlier detected in a scatter plot

some statistical methods available for detecting outliers. These methods can, for example, be based on the fact that the data comes from a normal distribution and determining the probability of finding a value such as the largest or smallest value from this distribution. This can, for example, be done by looking at the difference between possible outliers and the mean of all values or at the difference between the outlier and its closest value, and then determining the probability of finding as large a difference as was found. This study is conducted to evaluate if it is possible that the outlier found can come from the normal distribution, although it seems like an extreme value.

Notice that data reduction as discussed here is related to data validation as discussed in Chap. 10. Data validation deals with identifying false data points based on the execution of the experiment, such as determining if people have participated seriously in the experiment. The type of data reduction discussed in this section is concerned with identifying outliers not only based on the experiment execution, but instead looking at the results from the execution in the form of collected data and taking into account, for example, descriptive statistics.

When outliers have been identified, it is important to decide what to do with them. This should not only be based on the coordinates in the diagram; here it is important to analyze the reasons for the outliers. If the outlier is due to a strange or rare event that will never happen again, the point could be excluded. This can, for example, be the case if the point is completely wrong or misunderstood.

If the outlier is due to a rare event that may occur again, for example, if a module was implemented by inexperienced staff, it is not advisable to exclude the value from the analysis, because there is much relevant information in the outlier. If the outlier is due to a variable that was not considered before, such as the staff experience, it may be considered to base the calculations and models also on this variable. It is also possible to derive two separate models. In the case with staff experience it means one model based on normal staff (with the outlier removed) and one separate model for inexperienced staff. How to do this must be decided for every special case.

It is not only invalid data that can be removed from the data set. It is sometimes ineffective to analyze redundant data if the redundancy is too large. One way to identify redundant data is through factor analysis and principal component analysis (PCA). These techniques identify orthogonal factors that can be used instead of the original factors. It is beyond the scope of this book to describe these types of

techniques. The reader can refer instead to, for example, Kachigan [124, 125] and Manly [165].

11.3 Hypothesis Testing

11.3.1 Basic Concept

The objective of hypothesis testing is to see if it is possible to reject a certain null hypothesis, H_0, based on a sample from some statistical distribution. That is, the null hypothesis describes some properties of the distribution from which the sample is drawn and the experimenter wants to reject that these properties are true with a given significance. The null hypothesis is also discussed in Chap. 9. A common case is that the distribution depends on a single parameter. Setting up H_0 then means formulating the distribution and assigning a value to the parameter, which will be tested.

For example, an experimenter observes a vehicle and wants to show that the vehicle is not a car. The experimenter knows that all cars have four wheels, but also that there are other vehicles than cars that have four wheels. A very simple example of a null hypothesis can be formulated as "H_0: the observed vehicle is a car."

To test H_0, a test unit, t, is defined and a critical area, C, is given as well which is a part of the area over which t varies. This means that the significance test can be formulated as follows:

- If $t \in C$, reject H_0.
- If $t \notin C$, do not reject H_0.

In our example, the test unit t is the number of wheels and the critical area is $C = 1,\ 2,\ 3,\ 5,\ 6, \ldots$. The test is: if $t \leq 3$ or $t \geq 5$, reject H_0, otherwise do not reject H_0.

If it is observed that $t = 4$, it means that the hypothesis cannot be rejected and no conclusion can be drawn. This is because there may be other vehicles than cars with four wheels.

The null hypothesis should hence be formulated negatively, i.e., the intention of the test is to reject the hypothesis. If the null hypothesis is not rejected, nothing can be said about the outcome, while if the hypothesis is rejected, it can be stated that the hypothesis is false with a given significance (α) (see below). When a test is carried out it is in many cases possible to calculate the lowest possible significance (often denoted the p-value) with which it is possible to reject the null hypothesis. This value is often reported from statistical analysis packages.

The critical area, C, may have different shapes, but it is very common that it has some form of intervals, for example $t \leq a$ or $t \geq b$. If C consists of one such interval it is one-sided. If it consists of two intervals ($t \leq a$, $t \geq b$, where $a < b$), it is two-sided.

Three important probabilities concerning hypothesis testing are as follows:

$$\alpha = P(\text{type-I-error}) = P(\text{reject } H_0 \mid H_0 \text{ is true})$$

$$\beta = P(\text{type-II-error}) = P(\text{not reject } H_0 \mid H_0 \text{ is false})$$

$$\text{Power} = 1 - \beta = P(\text{reject } H_0 \mid H_0 \text{ is false})$$

These probabilities are also discussed in Chap. 9.

We try here to illustrate the concepts in a small example describing a simple but illustrative test called the *binomial test*. An experimenter has measured a number of failures during operation for a product and classified them as corruptive (faults that do corrupt the program's data) and non-corruptive (faults that do not corrupt the program's data). The experimenter's theory is that the non-corruptive faults are more common than the corruptive faults. The experimenter therefore wants to perform a test to investigate if the difference in the number of faults of the different types is due to coincidence or if it reveals a systematic difference.

The null hypothesis is that there is no difference in the probability of receiving a corruptive fault and receiving a non-corruptive fault. That is, the null hypothesis can be formulated as follows:

$$H_0 : P(\text{corruptive fault}) = P(\text{non-corruptive fault}) = 1/2$$

It is decided that α should be less than 0.10. The experimenter has received the following data:

- There are 11 faults that are non-corruptive.
- There are four faults that are corruptive.

If the null hypothesis is true, the probability of obtaining as few as four (i.e., four or fewer) corruptive faults out of 15 is

$$P(\text{0-4 corruptive faults}) = \sum_{i=0}^{4} \binom{15}{i} (\frac{1}{2})^i (\frac{1}{2})^{15-i} = \frac{1}{2^{15}} \sum_{i=0}^{4} \binom{15}{i} = 0.059$$

That is, if the experimenter concludes that the data that has been received shows that the non-corruptive faults are more common than the corruptive faults, the probability of committing a type-I-fault is 0.059. In this case the experimenter can reject H_0 because $0.059 < 0.10$.

The probability of receiving five or fewer corruptive faults, if the null hypothesis is true, can be computed to be 0.1509. This is larger than 0.10, which means that five corruptive faults out of 15 would not be sufficient to reject H_0. The experimenter

can therefore decide more formally to interpret the data in an experiment with 15 received faults as follows:

- If four or fewer faults are corruptive, reject H_0.
- If more than four faults are corruptive, do not reject H_0.

To summarize, the number of received corruptive faults (out of 15 faults) is the test unit and the critical area is 0, 1, 2, 3, and 4 (corruptive faults).

Based on this, it is interesting to determine the power of the formulated test. Since the power is the probability of rejecting H_0 if H_0 is not true, we have to formulate what we mean by H_0 is not true. In our example this can be formulated as follows:

$$P(\text{corruptive fault}) < P(\text{non-corruptive fault})$$

Since the sum of the two probabilities equals 1, this can also be formulated as follows:

$$P(\text{corruptive fault}) = a < 1/2$$

The probability of receiving four or fewer corruptive faults out of 15 faults (i.e., the probability of rejecting H_0 if H_0 is false) is as follows:

$$p = \sum_{i=0}^{4} \binom{15}{i} a^i (1-a)^{15-i}$$

This probability is plotted for different values of a in Fig. 11.9.

It can be seen that the power of the test is high if the difference between the probabilities of receiving a corruptive fault and a non-corruptive fault is large. If, for example, $a = 0.05$ there is a very great chance that there will be four or fewer corruptive faults. On the other hand, if the difference is very small, the power will be smaller. If, for example, $a = 0.45$ there is a great chance that there will be more than four corruptive faults.

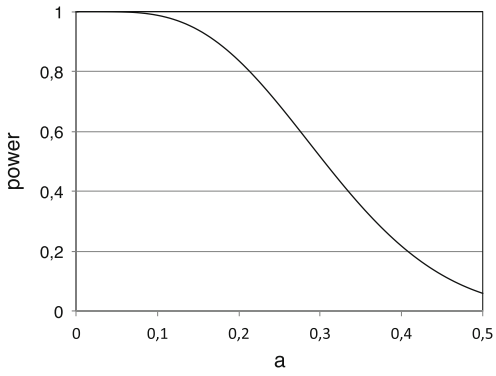


Fig. 11.9 Power of a one-sided binomial test

There are a number of factors affecting the power of a test. First, the test itself can be more or less effective. Second, the sample size affects the power. A larger sample size means higher power. Another aspect that affects the power is the choice of a one-sided or two-sided alternative hypothesis. A one-sided hypothesis gives a higher power than a two-sided hypothesis.

Power in software engineering experimentation is assessed and further discussed by Dybå et al. [64].

11.3.2 Parametric and Non-parametric Tests

Tests can be classified into parametric tests and non-parametric tests. Parametric tests are based on a model that involves a specific distribution. In most cases, it is assumed that some of the parameters involved in a parametric test are normally distributed. One test for normality is the Chi-square test, which is further described below when discussing the different types of tests. Parametric tests also require that parameters can be measured at least on an interval scale. If parameters cannot be measured on at least an interval scale this means generally that parametric tests cannot be used. In these cases there are a wide range of non-parametric tests available.

Non-parametric tests do not make the same type of assumptions concerning the distribution of parameters as do parametric tests. Only very general assumptions are made to derive non-parametric tests. The binomial test, described in the previous subsection, is an example of a non-parametric test. Non-parametric tests are more general than parametric tests. This means that non-parametric tests, if they are available, can be used instead of parametric tests, but parametric tests cannot generally be used when non-parametric tests can be used. With respect to the choice of parametric or non-parametric test, there are two factors to consider:

Applicability	What are the assumptions made by the different tests? It is important that assumptions regarding distributions of parameters and assumptions concerning scales are realistic.
Power	The power of parametric tests is generally higher than for non-parametric tests. Therefore, parametric tests require fewer data points, and therefore smaller experiments, than non-parametric tests if the assumptions are true.

The choice between parametric and non-parametric statistical methods is also discussed by Briand et al. [38]. They state that even if it is a risk using parametric methods when the required conditions are not fulfilled, it is in some cases worth taking that risk. Simulations have shown that parametric methods, such as the t-test, described below, are fairly robust to deviations from the preconditions (interval scale) as long as the deviations are not too large.

11.3.3 Overview of Tests

In addition to the binomial test introduced above, the following tests are described in this section:

t-test	The t-test is one of the most often used parametric tests. The test is used to compare two sample means. That is, the design is one factor with two treatments.
Mann-Whitney	This is a non-parametric alternative to the t-test.
F-test	This is a parametric test that can be used to compare two sample distributions.
Paired t-test	This is a t-test for a paired comparison design.
Wilcoxon	This is a non-parametric alternative to the paired t-test.
Sign test	This is a non-parametric alternative to the paired t-test. The sign test is a simpler alternative to the Wilcoxon test.
ANOVA	(ANalysis Of VAriance). This is a family of parametric tests that can be used for designs with more than two levels of a factor. ANOVA tests can, for example, be used in the following designs: one factor with more than two levels, one factor and blocking variable, factorial design, and nested design.
Kruskal-Wallis	This is a non-parametric alternative to ANOVA in the case of one factor with more than two treatments.
Chi-square	This is a family of non-parametric tests that can be used when data are in the form of frequencies.

The different tests can be sorted with respect to design type and with respect to whether they are parametric or non-parametric as in Table 11.3.

For all tests that are described, the following items are presented in separate tables for each test:

Input	The type of measurements needed to make the test applicable describes the input to the test. That is, this describes what

Table 11.3 Overview of parametric/non-parametric tests for different designs

Design	Parametric	Non-parametric
One factor, one treatment		Chi-square, Binomial test
One factor, two treatments, completely randomized design	t-test, F-test	Mann-Whitney, Chi-square
One factor, two treatments, paired comparison	Paired t-test	Wilcoxon, Sign test
One factor, more than two treatments	ANOVA	Kruskal-Wallis, Chi-square
More than one factor	ANOVA[a]	

[a] This test is not described in this book. Refer instead to, for example, Marascuilo and Serlin [166] and Montgomery [180].

	requirements there are on the experiment design if the test should be applicable.
Null hypothesis	A formulation of the null-hypothesis is provided.
Calculations	It describes what to calculate based on the measured data.
Criterion	The criterion for rejecting the null hypothesis. This often involves using statistical tables and it describes which table to use from Appendix B. In this book tables are only provided for one level of significance, but for many tests references are given to other sources where more comprehensive tables are provided.

All tests are not described completely here. For more information concerning the tests refer to the references given in the text. For example, the Mann-Whitney test, the Wilcoxon test, the sign test, and the Kruskal-Wallis test are described for the most straightforward case with few samples. If there are many samples (e.g., more than about 35 for the sign test) it is in many cases hard to do the calculations and make the decisions as described below. In these cases it is possible to do certain approximations because there are so many samples. How to do this is described by, for example, Siegel and Castellan [228]. They also described how to do this when ties (two or more equal values) occur for those tests.

The objective of the descriptions of the tests is that it should be possible to use the tests based on the descriptions and the examples. The intention is not to provide all details behind the derivations of the formulas.

Using the type of description outlined above, our simple example test (see Sect. 11.3.1) is summarized in Table 11.4.

In Table 11.4, the binomial test is described for the null hypothesis that the two events are equally probable. It is possible to state other null hypotheses, such as $P(\text{event 1}) = 0.3$ and $P(\text{event 2}) = 0.7$. For a description of how to perform the test in those cases, see, for example, Siegel and Castellan [228].

For most of the tests in this chapter, examples of usage are presented. The examples are based on fictitious data. Moreover, the tests are primarily presented for a significance level of 5% for which tables are provided in Appendix B. More comprehensive tables are available in books on statistics, for example, by Marascuilo and Serlin [166] and Montgomery [180].

Table 11.4 Binomial test

Item	Description
Input	Number of events counted for two different kinds of events (event_1 and event_2).
H_0	$P(\text{event 1}) = P(\text{event 2})$
Calculations	Calculate $p = \frac{1}{2^N}\sum_{i=0}^{n}\binom{N}{i}$ where N is the total number of events, and n is the number of the rarest event
Criterion	Two-sided ($H_1 : P(\text{event 1}) \neq P(\text{event 2})$): reject H_0 if $p < \alpha/2$
	One-sided ($H_1 : P(\text{event 1}) < P(\text{event 2})$): reject H_0 if $p < \alpha$ and event 1 is the rarest event in the sample

11.3.4 Tools for Data Analysis

Several statistical software tools are available, which include implementations of the tests, and much more functionality. There are both commercial and open source tools. One example of a commonly used open source tool is R.[1] An alternative is to use the Python programming language with packages such as SciPy[2] [263]. Tools of this type include functionality for many of the analysis tasks conducted by researchers. A large set of statistical tests and other functions are available, and they can often be used by researchers throughout the whole analysis process. In the two tools mentioned above, analysis is conducted by giving commands in a programming language, either as command-line instructions or in a script. They include functionality for importing data from various sources like comma separated files, databases, and spreadsheets. This makes it easy to import data. When data has been imported, it is possible to sort, filter, and select subsets of data. For example, if inspection data is analyzed there are commands for selecting subsets like data from a certain set of reviewers or data from reviews of certain artifacts. As mentioned above, these tools include functions for execution of a large amount of statistical test, and a number of other analyses like checking if data is normally distributed, correlation analysis, etc. There are also useful functions for producing graphical plots, like bar charts, box plots, and many other types of charts.

The amount of functionality in these types of tools is very large. In the "standard version" of the tool that is initially downloaded and installed, a large number of functions are included per default, and it is possible to extend the number by installing packages with additional functionality. It should be noted that it is important to understand the functions that are used, e.g., under which conditions they can be used, and how the results should be interpreted.

11.3.5 t-Test

The t-test is a parametric test used to compare two independent samples. That is, the design should be one factor with two levels. The t-test can be performed based on a number of different assumptions, but here an often-used alternative is described. For more information, refer, for example, to Montgomery [180], Siegel and Castellan [228], and Marascuilo and Serlin [166]. The test is performed as presented in Table 11.5.

[1] https://www.r-project.org/

[2] https://scipy.org/

Table 11.5 t-test

Item	Description
Input	Two independent samples: $x_1, x_2, \ldots x_n$ and $y_1, y_2, \ldots y_m$
H_0	$\mu_x = \mu_y$, i.e., the expected mean values are the same
Calculations	Calculate $t_0 = \dfrac{\bar{x} - \bar{y}}{S_p\sqrt{\frac{1}{n} + \frac{1}{m}}}$ where $S_p = \sqrt{\dfrac{(n-1)S_x^2 + (m-1)S_y^2}{n+m-2}}$ and S_x^2 and S_y^2 are the individual sample variances
Criterion	Two-sided ($H_1 : \mu_x \neq \mu_y$): reject H_0 if $\lvert t_0 \rvert > t_{\alpha/2,n+m-2}$. Here, $t_{\alpha,f}$ is the upper α percentage point of the t distribution with f degrees of freedom, which is equal to $n + m - 2$. The distribution is tabulated, for example, in Table B.1 and by Montgomery [180] and Marascuilo and Serlin [166]
	One-sided ($H_1 : \mu_x > \mu_y$): reject H_0 if $t_0 > t_{\alpha,n+m-2}$

Example of t-Test The defect densities in different programs have been compared in two projects. In one of the projects the result is

$$x = 3.42,\ 2.71,\ 2.84,\ 1.85,\ 3.22,\ 3.48,\ 2.68,\ 4.30,\ 2.49,\ 1.54$$

and in the other project the result is

$$y = 3.44,\ 4.97,\ 4.76,\ 4.96,\ 4.10,\ 3.05,\ 4.09,\ 3.69,\ 4.21,\ 4.40,\ 3.49$$

The null hypothesis is that the defect density is the same in both projects, and the alternative hypothesis is that it is not. Based on the data it can be seen that $n = 10$ and $m = 11$. The mean values are $\bar{x} = 2.853$ and $\bar{y} = 4.1055$.

It can be found that $S_x^2 = 0.6506$, $S_y^2 = 0.4112$, $S_p = 0.7243$, and $t_0 = -3.96$.

The number of degrees of freedom is $f = n + m - 2 = 10 + 11 - 2 = 19$. In Table B.1, it can be seen that $t_{0.025,19} = 2.093$. Since $\lvert t_0 \rvert > t_{0.025,19}$ it is possible to reject the null hypothesis with a two-tailed test at the 0.05 level.

11.3.6 Mann-Whitney

The Mann-Whitney test is a non-parametric alternative to the t-test. It is always possible to use this test instead of the t-test if the assumptions made by the t-test seem uncertain. The test, which is based on ranks, is not described completely here. More details are presented by, for example, Siegel and Castellan [228],[3] and Marascuilo and Serlin [166]. The test is summarized in Table 11.6.

[3] Siegel and Castellan [228] describe the Wilcoxon-Mann-Whitney test instead of the Mann-Whitney test. The two tests are, however, essentially the same.

Table 11.6 Mann-Whitney

Item	Description
Input	Two independent samples: $x_1, x_2, \ldots x_n$ and $y_1, y_2, \ldots y_m$
H_0	The two samples come from the same distribution
Calculations	Rank all samples and calculate $U = N_A N_B + \frac{N_A(N_A+1)}{2} - T$ and $U' = N_A N_B - U$, where $N_A = \min(n, m)$, $N_B = \max(n, m)$, and T is the sum of the ranks of the smallest sample
Criterion	Tables providing criterion for rejection of the null hypothesis based on the calculations are provided, for example, in Table B.3 and by Marascuilo and Serlin [166]
	Reject H_0 if $\min(U, U')$ is less than or equal to the value in Table B.3

Table 11.7 F-test

Item	Description
Input	Two independent samples: $x_1, x_2, \ldots x_n$ and $y_1, y_2, \ldots y_m$
H_0	$\sigma_x^2 = \sigma_y^2$, i.e., the variances are equal
Calculations	Calculate $F_0 = \frac{\max(S_x^2, S_y^2)}{\min(S_x^2, S_y^2)}$, where S_x^2 and S_y^2 are the individual sample variances
Criterion	Two-sided ($H_1 : \sigma_x^2 \neq \sigma_y^2$): reject H_0 if $F_0 > F_{\alpha/2, n_{max}-1, n_{min}-1}$, where n_{max} is the number of scores in the sample with maximum sample variance and n_{min} is the number of scores in the sample with minimum sample variance. $F_{\alpha/2, f_1, f_2}$ is the upper α percentage point of the F distribution with f_1 and f_2 degrees of freedom, which is tabulated, for example, in Table B.5 and by Montgomery [180], and Marascuilo and Serlin [166]
	One-sided ($H_1 : \sigma_x^2 > \sigma_y^2$): reject H_0 if $F_0 > F_{\alpha, n_{max}-1, n_{min}-1}$, and $S_x^2 > S_y^2$

Example of Mann-Whitney When the same data is used, as in the example with the t-test, it can be seen that $N_A = \min(10, 11) = 10$ and $N_B = \max(10, 11) = 11$. The ranks of the smallest sample (x) are 9, 5, 6, 2, 8, 11, 4, 17, 3, 1 and the ranks of the largest sample (y) are 10, 21, 19, 20, 15, 7, 14, 13, 16, 18, 12. Based on the ranks it can be found that $T = 66$, $U = 99$ and $U' = 11$. Since the smallest of U and U' is smaller than 26 (see Table B.3), it is possible to reject the null hypothesis with a two-tailed test at the 0.05 level.

11.3.7 F-Test

The F-test is a parametric test that can be used to compare the variance of two independent samples. More details about the test are presented by, for example, Montgomery [180], Robson [207], and Marascuilo and Serlin [166]. The test is performed as presented in Table 11.7.

Example of F-Test When the same data is used, as in the example with the t-test, it can be found that $S_x = 0.6506$ and $S_y = 0.4112$, which means that $F_0 = 1.58$. It can also be seen that $n_{max} = 10$ and $n_{min} = 11$.

From Table B.5 it can be seen that $F_{0.025,9,10} = 3.78$. Since $F_0 < F_{0.025,9,10}$ it is impossible to reject the null hypothesis with a two-tailed test at the 0.05 level. Thus, the test does not reject that the two samples have the same variance.

11.3.8 Paired t-Test

The paired t-test is used when two samples resulting from repeated measures are compared. This means that measurements are made with respect to, for example, a subject more than once. An example of this is if two tools are compared. If two groups independently use the two different tools, the result would be two independent samples and the ordinary t-test could be applied. If instead only one group were to be used and every person used both tools, we would have repeated measures. In this case the test examines the difference in performance for every person with the different tools.

The test, which is described in more detail by, for example, Montgomery [180] and Marascuilo and Serlin [166], is performed as presented in Table 11.8:

Example of Paired t-Test Ten programmers have independently developed two different programs. They have measured the effort that was required and the result is displayed in Table 11.9.

Table 11.8 Paired t-test

Item	Description
Input	Paired samples: $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$
H_0	$\mu_d = 0$, where $d_i = x_i - y_i$, i.e., the expected mean of the differences is 0
Calculations	Calculate $t_0 = \frac{\bar{d}}{S_d/(\sqrt{n})}$, where $S_d = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \bar{d})^2}{n-1}}$
Criterion	Two-sided ($H_1 : \mu_d \neq 0$): reject H_0 if $\lvert t_0 \rvert > t_{\alpha/2,n-1}$. Here, $t_{\alpha,f}$ is the upper α percentage point of the t distribution with f degrees of freedom. The distribution is tabulated, for example, in Table B.1 and by Montgomery [180] and Marascuilo and Serlin [166]
	One sided ($H_1 : \mu_d > 0$): reject H_0 if $\lvert t_0 \rvert > t_{\alpha,n-1}$.

Table 11.9 Required effort

Programmer	1	2	3	4	5	6	7	8	9	10
Program 1	105	137	124	111	151	150	168	159	104	102
Program 2	86.1	115	175	94.9	174	120	153	178	71.3	110

The null hypothesis is that the required effort to develop program 1 is the same as the required effort to develop program 2. The alternative hypothesis is that it is not. In order to carry out the test the following are calculated:

$d = \{18.9,\ 22,\ -51,\ 16.1,\ -23,\ 30, 15,\ -19,\ 32.7,\ -8\}$

$S_d = 27.358$

$t_0 = 0.39$

The number of degrees of freedom is $f = n - 1 = 10 - 1 = 9$. In Table B.1, it can be seen that $t_{0.025,9} = 2.262$.

Since $t_0 < t_{0.025,9}$ it is impossible to reject the null hypothesis with a two-sided test at the 0.05 level.

11.3.9 Wilcoxon

The Wilcoxon test is a non-parametric alternative to the paired t-test. The only requirements are that it is possible to determine which of the measures in a pair is the greatest and that it is possible to rank the differences. The test, which is based on ranks, is not described in detail here. A more detailed description is presented by, for example, Siegel and Castellan [228], and Marascuilo and Serlin [166]. The test is summarized in Table 11.10.

Example of Wilcoxon When the same data is used, as in the example with the paired t-test, it is found that the ranks of the absolute values of the difference (d) are 4, 6, 10, 3, 7, 8, 2, 5, 9, 1. Based on this T^+ and T^- can be calculated to be 32 and 23.

Since the smallest of T^+ and T^- is larger than 8 (see Table B.4) it is impossible to reject the null hypothesis with a two-tailed test at the 0.05 level.

Table 11.10 Wilcoxon

Item	Description
Input	Paired samples: $(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)$
H_0	If all differences ($d_i = x_i - y_i$) are ranked $(1, 2, 3 \ldots)$ without considering the sign, then the sum of the ranks of the positive differences equals the sum of the ranks of the negative differences
Calculations	Calculate T^+ as the sum of the ranks of the positive d_i:s and T^- as the sum of the ranks of the negative d_i:s
Criterion	Tables that can be used to determine if H_0 can be rejected based on T^+, T^- and the number of pairs, n, are available. See, for example, Table B.4 or Siegel and Castellan [228] and Marascuilo and Serlin [166] Reject H_0 if $\min(T^+, T^-)$ is less than or equal to the value in Table B.4

Table 11.11 Sign test

Item	Description
Input	Paired samples: $(x_1, y_1), (x_2, y_2) \ldots (x_N, y_N)$
H_0	$P(+) = P(-)$, where $+$ and $-$ represent the two events that $x_i > y_i$ and $x_i < y_i$
Calculations	Represent every positive difference ($d_i = x_i - y_i$) by a $+$ and every negative difference by a $-$. Calculate $p = \frac{1}{2^N} \sum_{i=0}^{n} \binom{N}{i}$, where N is the total number of signs, and n is number of signs of the rarest signs
Criterion	Two-sided ($H_1 : P(+) \neq P(-)$): reject H_0 if $p < \alpha/2$ One-sided ($H_1 : P(+) < P(-)$): reject H_0 if $p < \alpha$ and the $+$ event is the rarest event in the sample

11.3.10 Sign Test

The sign test is, like the Wilcoxon test, a non-parametric alternative to the paired t-test. The sign test can be used instead of the Wilcoxon test when it is not possible or necessary to rank the differences, since it is only based on the sign of the difference of the values in each pair. For example, it is not necessary to use the Wilcoxon test when it is possible to show significance with the sign test. This is due to the fact that the sign test has a lower power. Moreover, the sign test is easier to perform.

The test is further described by, for example, Siegel and Castellan [228] and Robson [207], and is summarized in Table 11.11.

The reader may recognize that this test is a binomial test where the two events are $+$ and $-$ respectively.

Example of Sign Test When the same data is used as in the example with the paired t-test it is found that there are six positive differences and four negative. This means that

$$p = \frac{1}{2^{10}} \sum_{i=0}^{4} \binom{10}{i} = \frac{193}{512} \approx 0.3770$$

Since $p > 0.025$ it is impossible to reject the null hypothesis with a two-tailed test at the 0.05 level.

11.3.11 ANOVA (ANalysis Of VAriance)

Analysis of variance can be used to analyze experiments from a number of different designs. The name, analysis of variance, is used because the method is based on looking at the total variability of the data and the variability partition according to

Table 11.12 ANOVA, one factor, more than two treatments

Item	Description
Input	a samples: $x_{11}, x_{12}, \ldots x_{1n_1}; x_{21}, x_{22}, \ldots x_{2n_2}; \ldots; x_{a1}, x_{a2}, \ldots x_{an_a}$
H_0	$\mu_{x_1} = \mu_{x_2} = \ldots = \mu_{x_a}$, i.e., all expected means are equal
Calculations	$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{n_i} x_{ij}^2 - \frac{x_{..}^2}{N}$
	$SS_{Treatment} = \sum_{i=1}^{a} \frac{x_{i.}^2}{n_i} - \frac{x_{..}^2}{N}$
	$SS_{Error} = SS_T - SS_{Treatment}$
	$MS_{Treatment} = SS_{Treatment}/(a-1)$
	$MS_{Error} = SS_{Error}/(N-a)$
	$F_0 = MS_{Treatment}/MS_{Error}$
	where N is the total number of measurements and a dot index denotes a summation over the dotted index, e.g., $x_{i.} = \sum_j x_{ij}$
Criterion	Reject H_0 if $F_0 > F_{\alpha,a-1,N-a}$. Here, F_{α,f_1,f_2} is the upper α percentage point of the F distribution with f_1 and f_2 degrees of freedom, which is tabulated, for example, in Table B.5 and by Montgomery [180] and Marascuilo and Serlin [166]

Table 11.13 ANOVA table for the ANOVA test described above

Source of variation	Sum of squares	Degrees of freedom	Mean square	F_0
Between treatments	$SS_{Treatment}$	$a-1$	$MS_{Treatment}$	$F_0 = \frac{MS_{Treatment}}{MS_{Error}}$
Error[a]	SS_{Error}	$N-a$	MS_{Error}	
Total	SS_T	$N-1$		

[a] This is sometimes denoted within treatments

different components. In its simplest form the test compares the variability due to treatment and the variability due to random error.

In this section, it is described how to use ANOVA in its simplest form. The test can be used to compare if a number of samples have the same mean value, that is, the design is one factor with more than two treatments. The test is summarized in Table 11.12

The results of an ANOVA test are often summarized in an ANOVA table. The results of a test for one factor with multiple levels can, for example, be summarized as in Table 11.13.

Notice that the described ANOVA test is just one variant of ANOVA tests. ANOVA tests can be used for a number of different designs, involving many different factors, blocking variables, etc. It is beyond the scope of this book to describe these tests in detail. The reader may refer instead to, for example, Montgomery [180] and Marascuilo and Serling [166].

Table 11.14 ANOVA table

Source of variation	Sum of squares	Degrees of freedom	Mean square	F_0
Between treatments	579.0515	2	289.5258	0.24
Error	36151	30	1205	
Total	36730	32		

Table 11.15 Kruskal-Wallis

Item	Description
Input	a samples: $x_{11}, x_{12}, \ldots x_{1n_1}; x_{21}, x_{22}, \ldots x_{2n_2}; \ldots; x_{a1}, x_{a2}, \ldots x_{an_a}$
H_0	The population medians of the a samples are equal
Calculations	All measures are ranked in one series $(1, 2, \ldots n_1 + n_2 + \ldots + n_a)$, and the calculations are based on these ranks. See, for example, [166, 228]
Criterion	See, for example, Siegel and Castellan [228] and Marascuilo and Serlin [166]

Example of ANOVA The module sizes in three different programs have been measured. The result is as follows:

- Program 1: 221, 159, 191, 194, 156, 238, 220, 197, 197, 194
- Program 2: 173, 171, 168, 286, 206, 140, 226, 248, 189, 208, 213
- Program 3: 234, 188, 181, 207, 266, 153, 190, 195, 181, 238, 191, 260

The null hypothesis is that the mean module size is the same in all three programs. The alternative hypothesis is that it is not. Based on the data above, the ANOVA table in Table 11.14 can be calculated.

The number of degrees of freedom is $f_1 = a - 1 = 3 - 1 = 2$ and $f_2 = N - a = 33 - 3 = 30$. In Table B.5, it can be seen that $F_{0.025,2,30} = 4.18$. Since $F_0 < F_{0.025,2,30}$ it is impossible to reject the null hypothesis at the 0.025 level.

11.3.12 Kruskal-Wallis

The Kruskal-Wallis one-way analysis of variance by ranks is a non-parametric alternative to the parametric one factor analysis of variance described above. This test can always be used instead of the parametric ANOVA if it is not certain that the assumptions of ANOVA are met. The test, which is based on ranks, is not described in detail here. More details are presented by, for example, Siegel and Castellan [228] and Marascuilo and Serlin [166].

The test is summarized in Table 11.15.

Table 11.16 Frequency table for module size (variables) of two systems (groups)

Module size	System 1	System 2
small	15	10
medium	20	19
large	25	28

11.3.13 Chi-square

Chi-square (sometimes denoted χ^2) tests can be performed in a number of different ways. All Chi-square tests are, however, based on the fact that data is in the form of frequencies. An example of frequencies for two systems with a number of modules could be that for system 1 there are 15 small modules, 20 medium modules, and 25 large modules, while for system 2 there are 10 small modules, 19 medium modules, and 28 large modules. This is summarized in Table 11.16.

In this case a Chi-square test could be performed to investigate if the distribution of small, medium, and large modules is the same for the two systems.

Chi-square tests can also be performed with one group of data, to investigate if one measured frequency distribution is the same as a theoretical distribution. This test can, for example, be performed to check if samples can be seen as normally distributed.

In Table 11.17, a Chi-square test is summarized, which can be used to compare if measurements from two or more groups come from the same distribution.

Example of Chi-square Test If a Chi-square test is performed on the data in Table 11.16 then Table 11.18 can be constructed.

The null hypothesis is that the size distribution is the same in both systems, and the alternative hypothesis is that the distributions are different. Based on the data the test statistic can be calculated to $X^2 = 1.12$. The number of degrees of freedom is $(r-1)(k-1) = 2*1 = 2$. In Table B.2, it can be seen that $\chi^2_{0.05,2} = 5.99$. Since $X^2 < \chi^2_{0.05,2}$ it is impossible to reject the null hypothesis at the 0.05 level.

Chi-square Goodness of Fit Test A Chi-square test can also be carried out to check if measurements come from a certain distribution, e.g., the normal distribution. In this case a goodness of fit test is performed according to Table 11.19

If the goodness of fit test is performed for a continuous distribution, the possible values that can be measured must be divided into intervals so that each interval can represent one value. This must, for example, be done for the normal distribution.

If the distribution of H_0 is completely specified (for example, $P(X = 1) = 2/3$, $P(X = 2) = 1/3$) then no parameters must be estimated from the measured data (i.e., $e = 0$). On the other hand, for example, if the null hypothesis only specifies that the values comply with a normal distribution, two parameters must be estimated. Both the mean value and the standard deviation of the normal distribution must be estimated, otherwise it is not possible to determine the values of the different expected values, E_i, for the intervals. Therefore, in this case, $e = 2$.

Table 11.17 Chi-square, k independent samples (groups)

Item	Description
Input	Data as frequencies for k groups
H_0	Measurements from the k groups are from the same distribution
Calculations	Create a contingency table. An example of a contingency table for two groups and three variables (i.e., the same dimensions as the data in Table 11.16) can be constructed as follows:

variable	group 1	group 2	combined
1	n_{11}	n_{12}	R_1
2	n_{21}	n_{22}	R_2
3	n_{31}	n_{32}	R_3
Total	C_1	C_2	N

In this table n_{ij} denotes the frequency for variable i and group j, C_i denotes the sum of the frequencies for group i, and R_i denotes the sum for variable i. N is the sum of all frequencies

Compute $X^2 = \sum_{i=1}^{r}\sum_{j=1}^{k} \frac{(n_{ij} - E_{ij})^2}{E_{ij}}$ where $E_{ij} = \frac{R_i C_j}{N}$ (the expected frequency if H_0 is true), r is the number of variables and k is the number of groups.

Criterion	Reject H_0 if $X^2 > \chi^2_{\alpha,f}$, where f is the number of degrees of freedom determined as $f = (r-1)(k-1)$. $\chi^2_{\alpha,f}$ is the upper α percentage point of the Chi-square distribution with f degrees of freedom, which is tabulated, for example, in Table B.2 and by Siegel and Castellan [228]

Table 11.18 Calculations for Chi-square test. (Expected values, E_{ij}, are displayed in parentheses)

Module size	System 1	System 2	Combined
small	15 (12.8205)	10 (12.1795)	$R_1 = 25$
medium	20 (20)	19 (19)	$R_2 = 39$
large	25 (27.1795)	28 (25.8205)	$R_3 = 53$
Total	$C_1 = 60$	$C_2 = 57$	$N = 117$

Example: Chi-square Goodness of Fit Test for Normal Distribution Sixty students have developed the same program and the measured sizes are displayed in Table 11.20.

The null hypothesis is that the data is normally distributed, and the alternative hypothesis is that it is not. Based on the data, the mean and the standard deviation can be estimated: $x = 794.9833$ and $s = 83.9751$.

The range can be divided into segments which have the same probability of including a value if the data is actually normally distributed with mean x and standard deviation s. In this example the range is divided into 10 segments. In order to find the upper limit (x) of the first segment the following equation should be solved:

Table 11.19 Chi-square, goodness of fit

Item	Description
Input	Data as frequencies for one group (i.e., $O_1, O_2, \ldots O_n$, where O_i represents the number of observations in category i). Compare with Table 11.2.
H_0	Measurements are from a certain distribution
Calculations	Compute $X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$, where E_i is the expected number of observations in category i if H_0 is true and n is the number of categories
Criterion	Reject H_0 if $X^2 > \chi^2_{\alpha,f}$, where f is the number of degrees of freedom determined as $f = n - e - 1$, and e is the number of parameters that must be estimated from the original data (see below). $\chi^2_{\alpha,f}$ is the upper α percentage point of the Chi-square distribution with f degrees of freedom, which is tabulated, for example, in Table B.2 and by Siegel and Castellan [228]. This is a one-sided test

Table 11.20 Measured sizes in lines of code (LOC)

757	758	892	734	800	979	938	866	690	877	773	778
679	888	799	811	657	750	891	724	775	810	940	854
784	843	867	743	816	813	618	715	706	906	679	845
708	855	777	660	870	843	790	741	766	677	801	850
821	877	713	680	667	752	875	811	999	808	771	832

$P(X < x) = 1/10$ where X is N($\bar{x}$, s), which in terms of the standard normal distribution corresponds to

$P(X_s < (x - \bar{x})/s) = 1/10$ where X_s is N(0, 1), which is the same as

$P(X_s < z) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2/\pi}} e^{(-y^2)/2} dy = 1/10$ where X_s is N(0, 1) and $x = sz + \bar{x}$.

These equations can be solved in a number of different ways. One way is to iterate and use a computer to help find z or x. Another way is to use a table of the standard normal distribution (which shows $P(X_s < z)$ for different values of z). This type of table is available in most books on statistics. It is also possible to use a specialized table that directly shows the limit values for the segments, i.e., the values of z. This type of table is presented by Humphrey [113].

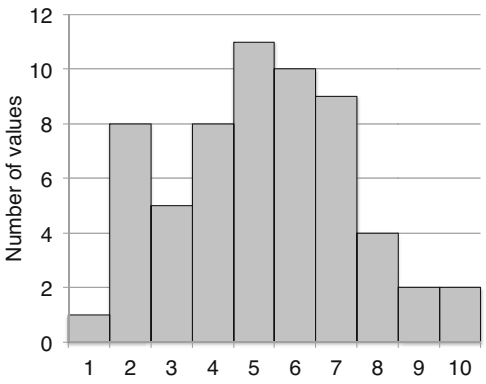
The resulting segment boundaries and the number of values that fall into each segment are shown in Table 11.21.

The expected number of values (E_i) in each segment is $60/10 = 6$. This means that $X^2 = 7.3$, i.e., $\frac{(8-6)^2}{6} + \frac{(6-6)^2}{6} + \ldots + \frac{(5-6)^2}{6}$. The number of degrees of freedom is $10 - 2 - 1 = 7$. In Table B.2, it can be seen that $\chi^2_{0.05,7} = 14.07$. Since $X^2 < \chi^2_{0.05,2}$, i.e., 7.3 is less than 14.07, it is impossible to reject the null hypothesis at the 0.05 level. If we look at a histogram (see Fig. 11.10) of the data, we can see that it seems to be fairly normally distributed.

Table 11.21 Segments

Segment	Lower boundary	Upper boundary	Number of values
1		687.3	8
2	687.3	724.3	6
3	724.3	750.9	4
4	750.9	773.7	6
5	773.7	795	5
6	795	816.3	9
7	816.3	839	2
8	839	865.7	6
9	865.7	902.6	9
10	902.6		5

Fig. 11.10 Histogram



Chi-square Test Final Remarks The Chi-square test is based on certain assumptions which are likely to be fulfilled if the expected values, E_i, are not too small. A rule of thumb is that if the number of degrees of freedom (f) is equal to 1, the Chi-square test should not be used if any of the expected frequencies are less than 5. If $f > 1$ the Chi-square test should not be used if more than 20% of the expected frequencies are less than 5 or any of them are less than 1. It should be observed that sometimes the test is used although the expected frequencies are not fulfilled. In these cases, this is a calculated risk.

One way to obtain larger expected frequencies is to combine related categories with new categories. However, the new categories must be meaningful. For more information concerning the Chi-square test, refer, for example, to Siegel and Castellan [228].

11.3.14 Model Adequacy Checking

Every statistical model relies on specific assumptions regarding, for example, distribution, independence, and scales. If the assumptions are invalidated by the data set, the results of the hypothesis testing are invalid. Hence, it is crucial to check that all assumptions are fulfilled.

The checking of model adequacy is done depending on the assumptions. Below we describe three cases:

Normality	If a test assumes that the data is normally distributed, a Chi-square test can be conducted to assess to which degree the assumption is fulfilled. The Chi-square test is described above.
Independence	If the test assumes that the data is a sample from several independent stochastic variables, it is necessary to check that there is no correlation between the sample sets. This may be checked with scatter plots and by calculating correlation coefficients as discussed at the beginning of this section.
Residuals	In many statistical models, there is a term that represents the residuals (statistical error). It is often assumed that the residuals are normally distributed. A common way to check this property is to plot the residuals in a scatter plot and see that there are no specific trends in the data (the distribution looks random).

11.3.15 Drawing Conclusions

When the experiment data has been analyzed and interpreted, we need to draw conclusions regarding the outcome of the experiment. If the hypotheses are rejected we may draw conclusions concerning the influence of the independent variables on the dependent variables, given that the experiment is valid (see Chap. 9).

If, on the other hand, the experiment cannot reject the null hypothesis, we cannot draw any conclusions on the independent variables' influence on the dependent variable. The only thing we have shown, in this case, is that there is no statistically significant difference between the treatments. However, there could have been, for this study, with more subjects participating, or, of course, in a new experiment.

If we have found statistically significant differences, we want to make general conclusions about the relation between independent and dependent variables. Before this can be done we need to consider the external validity of the experiment (see Chap. 9). We can only generalize the result to environments that are similar to the experimental setting.

Although the result of the experiment may be statistically significant, it is not necessarily the case that the result is of any practical importance. Assume, for example, that method X was shown with a high statistical significance to be 2%

more cost-effective than method Y; although there is a high statistical significance, the improvement of changing from method Y to method X might not be cost-effective. That is, it is necessary to study the observed effect size of different treatments and based on that draw conclusions and present recommendations. Kampenes et al. [128] give an overview of different effect size concepts and present a systematic review on how this is handled in published articles.

It may also be the opposite; although the experiment results may not be statistically significant or have a low statistical significance, the lessons learned from the experiment may still be of practical importance. The fact that a null hypothesis cannot be rejected with a certain level of significance does not mean that the null hypothesis is true. There may be problems with the design of the experiment, such as real threats to validity or too few data samples. Furthermore, depending on the situation and objective of the study, we may settle for a lower statistical significance since the results are of high practical importance. This issue is also related to the discussion concerning the priorities of different types of threats to validity (see Sect. 9.9).

When finding a significant correlation between a variable A and a variable B, we cannot, in general, draw the conclusion that there is a *causal relation* between A and B. There may be a third factor, C, that causes the measurable effects on A and B.

The conclusions drawn based on the outcome of the experiment are input to a decision, e.g., that a new method will be applied in future projects, or that further experimentation is needed.

It should be noted that there are drawbacks of using hypothesis testing too. As Miller [174] points out, most null hypotheses are formulated in such a way that they always will be rejected, if enough data points are provided, and it is not possible to actually obtain a sample that is representative of the whole population, for example, of all software engineers in the world. Care should always be taken when actions are taken based on the results of an experiment, and the experiment result should be seen as one factor in the decision process.

11.4 Example Analysis

The example is a continuation of the example in Sect. 10.4. Based on the experiment data from the execution, the first sub-step is to apply descriptive statistics, i.e., to plot the data. Appropriate statistical methods should be used in relation to the measurement scales, as described in Sect. 11.1. A commonly used way to plot data is to use box plots. They provide an excellent opportunity to get an overview of the data and to identify outliers. If identifying an outlier, it is important to understand whether there is some underlying explanation. For example, it may be the case that one or a few subjects have a very different background than the others, and hence it must be ensured that their data is comparable to the data from the other subjects. It may be particularly critical if only one of the groups is affected. In general, we

should be restrictive in removing data points, i.e., any removal of data should be well motivated and documented.

Once it is decided which data to include in the data analysis, it is time to take a look at the statistical analysis. The statistical analysis is always a challenge, and there are many different opinions about the use of different statistical methods and when to use parametric and non-parametric methods.

The first concern is to check if the data is normally distributed, for example by plotting a histogram (Fig. 11.5) or by using the Chi-square test as described in Sect. 11.3.13 or by using other alternative tests such as the so-called Kolmogorov-Smirnov test, the Shapiro-Wilks' W test, or the Anderson-Darling test. However with a small sample size, the data may look normally distributed without actually being normally distributed and normality tests may not detect it due to having few data points. Some parametric tests are more robust than others against deviations from normality. For example, the t-test is quite robust for non-normality, which is not the case for the ANOVA. Independently, it may be good to investigate whether the data is normally distributed.

Given the two-factor design (reading technique and requirements document) with two treatments each, there is a great need for the data to be normally distributed. If the data is normally distributed it is possible to use an ANOVA. If the data is not normally distributed there is a problem, since there is no non-parametric counterpart for this type of design, as illustrated in Table 11.3; if there is only one factor with two treatments, there are non-parametric alternatives. Thus, even if there are simpler designs, the chosen design seems quite straightforward and it is tempting to use it. However, it may not be a good choice to use this type of design. It generates more data points, but it does result in some challenges when it comes to the statistical analysis. Thus, it is important to be aware of the consequences in terms of analysis when selecting the experiment design. This type of design is sometimes referred to as a crossover design, i.e., first subjects use or are exposed to one treatment and then they are exposed to a second treatment. Some of the challenges would be addressed if it was possible to only have one factor, i.e., the reading technique. However, it is not realistic to use the same requirements document for two inspections unless there is a long time between the two runs. Kitchenham et al. [138] present some statistical challenges related to crossover design. Having said that, crossover designs are not uncommon in software engineering since it is a trade-off between the statistical challenges and having (too) few subjects assigned to each treatment, although others argue that crossover designs cannot be recommended in software engineering [138].

If we assume that the data is normally distributed, then an ANOVA test can be applied. However, a challenge is that if the ANOVA shows a significant result, it is still not known which difference is significant. To do this an additional test has to be used after the ANOVA, for example, Fisher's Protected Least Significant Difference (PLSD) test [180]. The test requires a significant ANOVA to be used, i.e., it is protected by a significant ANOVA. Fisher's PLSD is used to make a pairwise comparison of the means. Once again it illustrates some of the statistical challenges that come as a direct consequence of the experiment design. Thus, there is a need to have quite simple experiment designs to be able to make a correct statistical analysis.

If we had chosen to divide the subjects into two groups and then assigned them to use either PBR or CBR on one and the same requirements document, we would only have had one factor with two treatments. This means that a t-test or Mann-Whitney test could have been used depending on the outcome of the test for normality. On the other hand, we would have had no indication of the interaction between subject and treatment. Whether this is better or worse than other alternative designs has to be decided in each individual case depending on the number of subjects and the validity threats identified.

11.5 Exercises

11.1 What is descriptive statistics and what is it used for?

11.2 What is a parametric and non-parametric test respectively, and when can they be applied?

11.3 What is the power of a test?

11.4 What is a paired comparison?

11.5 Explain the ANOVA test briefly.

Chapter 12
Presentation and Package

When an experiment is completed, the findings may be presented for different audiences. This could, for example, be done in a paper for a conference or a journal, a report for decision-makers, a package for replication of the experiment, or as educational material. The packaging could also be done within companies to improve and understand different processes. In this case, it is appropriate to store the experiences in an experience base according to the concepts discussed by Basili et al. [29]. However, here we focus on the academic reporting in journals and conferences. If space limitations prevent complete reporting of all details, we recommend a technical report be published in parallel. The presentation and package step includes only one sub-step as illustrated in Fig. 12.1.

Jedlitschka and Pfahl proposed a scheme for the academic reporting of experiments [119] which was later evaluated by Kitchenham et al. [140]. Jedlitschka and Pfahl's proposal is summarized in Table 12.1 and briefly elaborated in Sect. 12.1.

12.1 Experiment Report Structure

Structured Abstract The abstract should give the reader a quick summary of the key characteristics of the experiment. Structured abstracts are empirically demonstrated to be efficient tools to aid extraction of data [42] as well as writing good abstracts [43]. The elements of a structured abstract are as follows:

- Background or Context
- Objectives or Aims
- Method
- Results
- Conclusions

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_12

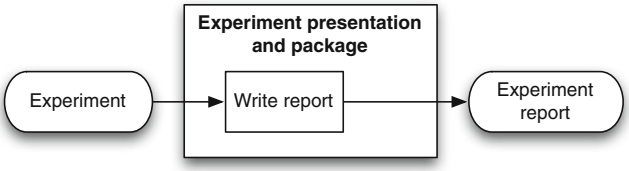


Fig. 12.1 Overview of presentation and package

Table 12.1 Proposed reporting structure for experiment reports, by Jedlitschka and Pfahl [119]

Sections/Subsections	Contents
Title, Authorship	
Structured abstract	Summarizes the paper under headings of Background or Context, Objectives or Aims, Method, Results, and Conclusions
Motivation	Sets the scope of the work and encourages readers to read the rest of the paper
Problem statement	Reports what the problem is; where it occurs, and who observes it
Research objectives	Defines the experiment using the formalized style used in GQM
Context	Reports environmental factors such as settings and locations
Related work	How current study relates to other research
Experimental design	Describes the outcome of the experimental planning stage
Goals, hypotheses, and variables	Presents the refined research objectives
Design	Define the type of experimental design
Subjects	Defines the methods used for subject sampling and group allocation
Objects	Defines what experimental objects were used
Instrumentation	Defines any guidelines and measurement instruments used
Data collection procedure	Defines the experimental schedule, timing and data collection procedures
Analysis procedure	Specifies the mathematical analysis model to be used
Evaluation of validity	Describes the validity of materials, procedures to ensure participants keep to the experimental method, and methods to ensure the reliability and validity of data collection methods and tools
Execution	Describes how the experimental plan was implemented
Sample	Description of the sample characteristics
Preparation	How the experimental groups were formed and trained
Data collection performed	How data collection took place and any deviations from plan
Validity procedure	How the validity process was followed and any deviation from plan
Analysis	Summarizes the collected data and describes how it was analyzed
Descriptive statistics	Presentation of the data using descriptive statistics

(continued)

Table 12.1 (continued)

Sections/Subsections	Contents
Data set reduction	Describes any reduction of the data set, e.g., removal of outliers
Hypothesis testing	Describes how the data was evaluated and how the analysis model was validated
Interpretation	Interprets the findings from the Analysis section
Evaluation of results and implications	Explains the results
Limitations of study	Discusses threats to validity
Inferences	How the results generalize given the findings and limitations
Lesson learnt	Descriptions of what went well and what did not during the course of the experiment
Conclusions and future work	Presents a summary of the study
Relation to existing evidence	Describes the contribution of the study in the context of earlier experiments
Impact	Identifies the most important findings
Limitations	Identifies main limitations of approach, i.e., circumstances when the expected benefits will not be delivered
Future work	Suggestions for other experiments to further investigate
Acknowledgements	Identifies any contributors who do not fulfill authorship criteria
References	Lists all cited literature
Appendices	Includes raw data and/or detailed analyses which might help others to use the results

Example An example of a structured abstract is presented to illustrate the items. In this case, the length of the structured abstract is limited to 300 words:

Context: Throughout an organization, people have different responsibilities and work tasks; hence, it is probable that different roles have different priorities when it comes to what should be improved within a company. This has been found in previous studies in marketing, but is this true for software improvement as well? *Objective:* This paper evaluates how different roles in a software development organization view different issues in software process improvement, and if such differences could be used in order to provide more tailor-made process improvements within an organization, and uses this as a working hypothesis. *Method:* A quantitative questionnaire containing five different weighted questions related to software process improvement was developed. Eighty-four employees from all levels of a Swedish telecommunication company were then approached, of which 63 responded. *Results:* The different roles disagreed on three of the questions while they agreed on two of the questions. The disagreement was related to issues about importance of improvement, urgency of problems, and threat against successful process management, while the questions where the roles agreed focused on communication of the processes (documentation and teaching). *Conclusion:* It is concluded that it is important to be aware of and take into account the different needs of different roles. This will make it possible to provide improvements

tailored to specific roles, which will probably help to overcome resistance to process improvements. It is also important to look into other areas and companies (for example, marketing) where it could be beneficial when conducting process improvements.

Motivation The motivation or introduction sets the scope and defines the objective of the research; hence it primarily reports the outcome of the scoping step (see Chap. 8). Information about the intent of the work can also be included to clarify and capture the reader's interest. This provides the reader with an understanding of why the research has been carried out and why there is a need for it. The context in which the experiment is conducted should be briefly presented here.

Related Work Related work is important to provide a picture of how the current experiment is related to work conducted previously. Every experiment report does not need a complete systematic literature review (see Chap. 4), although being systematic in searching for literature is mostly beneficial. In particular, in the case of replication studies, all previous studies should be reported.

Experimental Design Here, the outcome of the planning step is reported (see Chap. 9). The hypotheses which are derived from the problem statement are described in detail. The experimental design is presented, including the design type, variables measured, both independent and dependent, as well as the instrumentation.

A description of how the data will be collected and analyzed should be included. Moreover, a characterization of the subjects should be provided. The discussion about the experiment's conclusion and internal, construct, and external validity should be provided here together with the possible threats against the plans.

The purpose for describing these items is to enable other persons to both understand the design so that it is clear to the reader that the results are trustworthy and to enable replication of the study. In short, it should help the reader to get a deeper understanding of what has been done.

Execution The first part to describe is how the operation is prepared (see Chap. 10). It is important to include descriptions of aspects that will ease replication of the experiment and to give insight into how activities have been carried out. The preparation of the subjects has to be presented. Information such as whether they attended some lessons or not is important to provide. The execution of the experiment should also be presented and how data was collected during the experiment.

Validation procedures of the data collection are another issue that has to be stressed and it has to be reported if deviations from the plan have been made. All information is aimed to provide a case for the validity of the data and to highlight problems.

Analysis A presentation of the data analysis, where the calculations are described together with the assumptions for using some specific analysis model, should be provided. Information about sample sizes, significance levels, and application of tests must also be included so that the reader will know the prerequisites for the

analysis. The reasons for the actions taken, for example outlier removal, should be described to avoid misunderstandings in the interpretation of the results. For more information see Chap. 11.

Interpretation Raw results from the analysis are not enough to provide an understanding of the results and conclusions from the experiment. An interpretation must also be provided (see Chap. 11). It includes the rejection of the hypothesis or the inability to reject the null hypothesis. The interpretation summarizes how the results from the experiment may be used.

The interpretation should be done with references to validity (see Chap. 9). Factors that might have had an impact on the results should be described.

Conclusions and Further Work Finally, the discussions about the findings and the conclusions are presented as a summary of the whole experiment together with the outcomes, problems, deviations from the plans, and so forth. The results should also be related to work reported previously. It is important to address similarities and differences in the findings.

Ideas for future work might also be included in this section and information about where more information can be found to get a deeper insight into the experiment and to ease replication of the experiment.

Appendices Information that is not vital for the presentation could be included in appendices. This could, for example, be the collected data and more information about the subjects and objects. If the intention is to produce a lab package, the material used in the experiment could be provided here.

12.2 Exercises

12.1 Why is it important to document an experiment thoroughly?

12.2 What is a lab package? Can you find any lab packages on the Internet? What are potential concerns with lab packages?

12.3 Why is it important to report related work?

12.4 Why is it not enough just to provide the results from the analysis? In other words, why is a special interpretation of the results important?

12.5 Which information in the report is most essential when replicating an experiment?

Part III
Example Experiments

Chapter 13
Experiment Process Illustration

The primary objective of the presentation of this experiment is to illustrate experimentation and the steps in the experiment process introduced in the previous chapters. The presentation of the experiment in this chapter is focused on the experiment process rather than following the proposed report structure in Chap. 12.

The objective of the presented experiment is to investigate the performance in using the Personal Software Process (PSP) [113, 114] based on the individual backgrounds of the people taking the PSP course. The experiment, as reported here, is part of a larger investigation of the individual differences in performance within the PSP. Since "individual background" cannot be randomly assigned to subjects, the experiment is in fact a quasi-experiment.

The PSP is an individual process for a systematic approach to software development. The process includes, for example, measurement, estimation, planning, and tracking. Furthermore, reuse is a key issue, and in particular the reuse of individual experiences and data. The PSP course introduces the process in seven incremental steps adding new features to the process using templates, forms, and process scripts.

For the sake of simplicity, only two hypotheses are evaluated here. The data set for the larger study can be found in Appendix A.1.2. The experiment presented in this chapter uses a subset of the data.

13.1 Scoping

13.1.1 Goal Definition

First, it should be decided if an experiment is a suitable way to analyze the problem at hand. In this particular case, the objective of the empirical study is to determine

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_13

the differences in individual performance for people using the PSP given their background.

The experiment is motivated by a need to understand the differences in individual performance within the PSP. It is well known, and accepted, that software engineers perform differently. One objective of introducing a personal process is to provide support for the individuals to improve their performance. In order to support improvement in the best possible way, it is important to understand what differences still can be expected within the PSP and if it is possible to explain and thus understand the individual differences.

Object of Study The object of study is the participants in the Personal Software Process (PSP) course and their ability in terms of performance based on their background and experience. Humphrey defines the Personal Software Process in his two books on the subject [113, 114].

Purpose The purpose of the experiment is to evaluate the individual performance based on the background of the people taking the PSP course. The experiment provides insight into what can be expected in terms of individual performance when using the PSP.

Perspective The perspective is from the point of view of the researchers and teachers, i.e., the researcher or teacher would like to know if there are any systematic differences in the performance in the course based on the background of the individuals entering the PSP course. This also includes people who may want to take the course in the future or to introduce the PSP in industry.

Quality Focus The main effect studied in the experiment is the individual performance in the PSP course. Here, two specific aspects are emphasized. The choice is to focus on Productivity (KLOC/development time) and Defect density (faults/KLOC), where KLOC stands for thousands of lines of code.

Context The experiment is run within the context of the PSP. Moreover, the experiment is conducted within a PSP course given at the Department of Communication Systems, Lund University, Sweden. This study is from the course given in 1996–1997, and the main difference from the PSP as presented by Humphrey [113] is that it was decided to use a coding standard and a line counting standard. Moreover, the course was run with C as a mandatory programming language independently of the background of the students. The experimental context characterization is "multi-test within object study" (see Table 8.1). The study is focused on the PSP or more specifically the 10 programs in Humphrey's book [113] denoted 1A–10A. The PSP course is taken by a large number of individuals (this particular year, 65 students finished the course). Thus, 65 subjects are included in the study (see Sect. 8.1). Thus, from this definition the study can be judged as being a controlled experiment. The lack of randomization of students, i.e., the students signed up for the course, means, however, that the study still lacks one important ingredient to make it a fully controlled experiment. This is hence classified as a quasi-experiment (see Sect. 8.1).

13.1.2 Summary of Scoping

The summary is made according to Sect. 8.2.

Analyze *the outcome of the PSP*
for the purpose *of evaluation*
with respect to *the background of the individuals*
from the point of view of the *researchers and teachers*
in the context of *the PSP course.*

13.2 Planning

13.2.1 Context Selection

The context of the experiment is a PSP course at the university, and hence the experiment is run offline (not in an industrial software development), it is conducted by graduate students (normally students in their fourth year at the university), and it is specific since it is focused on the PSP in an educational environment. The ability to generalize from this specific context is further elaborated below when discussing threats to the validity in the experiment. The experiment addresses a real problem, i.e., the differences in individual performance and the understanding of the differences.

The use of the PSP as an experimental context provides other researchers with excellent opportunities to replicate the experiment as it is well defined. Furthermore, it means that there is no need to spend much effort in setting up the experiment in terms of defining and creating the environment in which the experiment is run. Humphrey [113] defines the experimental context, and hence there is no need to prepare forms for data collection and so forth.

13.2.2 Hypothesis Formulation

An important aspect of experiments is to know and to formally state clearly what is going to be evaluated in the experiment. This leads to the formulation of a hypothesis (or several hypotheses). Here, it has been decided to focus on two hypotheses. Informally, they are as follows:

1. Students from both the Computer Science and Engineering program and the Electrical Engineering program have taken the course. The students from the Computer Science and Engineering program normally have taken more courses in computer science and software engineering, and hence it is expected that

they have a higher productivity than students from the Electrical Engineering program.
2. As part of the first lecture, the students were asked to fill out a survey regarding their background in terms of experiences from issues related to the course. This can be exemplified with, for example, knowledge in C. The students were required to use C in the course independently of their prior experience of the language. Thus, it was not required that the students had taken a C course prior to entering the PSP course, which meant that some students learned C within the PSP course. This is not according to the recommendation by Humphrey [113]. The hypothesis based on the C experience is that students with more experience in C make fewer faults per line of code.

Based on this informal statement of the hypotheses, it is now possible to state them formally and also to define what measures are needed to evaluate the hypotheses.

1. Null hypothesis, H_0: There is no difference in productivity (measured as lines of code per total development time) between students from the Computer Science and Engineering program (CSE) and the Electrical Engineering program (EE).
   H_0: Prod(CSE) = Prod(EE).
   Alternative hypothesis, H_1: Prod(CSE) $\neq$ Prod(EE).
   Measures needed: student program (CSE or EE) and productivity (LOC/hour).
2. Null hypothesis, H_0: There is no difference between the students in terms of number of faults per KLOC (1000 lines of code) based on the prior knowledge in C.
   H_0: Number of faults per KLOC is independent of C experience.
   Alternative hypothesis, H_1: Number of faults per KLOC changes with C experience.
   Measures needed: C experience and Faults/KLOC.

The hypotheses mean that the following data needs to be collected:

- Student program: measured by CSE or EE (nominal scale).
- Productivity is measured as LOC/Development time. Thus, program size has to be measured (lines of code according to the coding standard and the counting standard) and development time (minutes spent developing the program). The development time is translated to hours when the productivity is calculated. It should be noted that it was chosen to study the total program size (the sum of the 10 programming assignments) and the development time for all 10 programs. Thus, the individual assignments are not studied. The following is counted:
  - Lines of code are measured by counting the lines of code using a line counter program (ratio scale). The lines counted are new and changed lines of code.
  - Development time is measured in minutes (ratio scale).
  - Productivity is hence measured on a ratio scale.

- C experience is measured by introducing a classification into four classes based on prior experience of C. The classes are:

  1. No prior experience
  2. Read a book or followed a course
  3. Some industrial experience (less than 6 months)
  4. Industrial experience (more than 6 months)

  The experience in C is hence measured on an ordinal scale.
- Faults/KLOC is measured as the number of faults divided by the number of lines of code.

The hypotheses and measures put constraints on the type of statistical test to use, at least formally. The measurement scales formally determine the application of specific statistical methods, but we may want to relax these requirements for other reasons. This issue is further discussed below, when discussing the actual type of design in the experiment.

13.2.3 Variables Selection

The independent variables are student program and experience in C. The dependent variables are productivity and faults/KLOC.

13.2.4 Selection of Subjects

The subjects are chosen based on convenience, i.e., the subjects are students taking the PSP course. The students are a sample from all students in the two programs, but not a random sample.

13.2.5 Experiment Design

The problem has been stated, and the independent and dependent variables have been chosen. Furthermore, the measurement scales have been decided for the variables. Thus, it is now possible to design the experiment. Thus, the general design principles for an experiment need to be addressed:

Randomization The object is not assigned randomly to the subjects. All students use the PSP and its 10 assignments. The objective of the study is not to evaluate the PSP vs. something else. The subjects are, as stated above, not selected randomly; they are the students that have chosen to take the course. Moreover, the assignments

are not made in random order. The order is, however, not important, since the measures used in the evaluation are the results of developing the 10 programs.

Blocking No systematic approach to blocking is applied. The decision to measure the 10 programs and evaluate based on this, rather than looking at each individual program, can be viewed as providing blocking for differences between the 10 programs, thus blocking the impact of the differences between individual programs.

Balancing It would have been preferable to have a balanced data set, but the experimental study is based on a course for which the participants have signed up, and hence it is impossible to influence the background of people and consequently to balance the data set.

Standard Design Types The information available is compared with the standard types of designs outlined in Chap. 9. Both designs can be found among the standard types, and the statistical tests are available in this book.

1. The definition, hypotheses, and measures for the first evaluation mean that the design is one factor with two treatments. The factor is the program and the treatments are CSE or EE. The dependent variable is measured on a ratio scale, and hence a parametric test is suitable. In this particular case, the t-test will be used.
2. The second design is of the type "one factor with more than two treatments." The factor is the experience in C with four treatments; see the experience grading above. The dependent variable is measured on a ratio scale and it is possible to use a parametric test for this hypothesis too. The ANOVA test is hence suitable to use for evaluation.

13.2.6 Instrumentation

The background and experience of the individuals are found through a survey handed out at the first lecture (see Table A.1 in Appendix A). This data provides the input to the characterization of the students, and hence are the independent variables in the experiment. The objects are the programs developed within the PSP course. The guidelines and measurements are provided through the PSP [113].

13.2.7 Validity Evaluation

There are four different types of validity threat to consider. *Internal validity* is primarily focused on the validity of the actual study. *External validity* can be divided into PSP students at Lund University in forthcoming years, students at Lund University (or more realistically students in the CSE and EE programs),

the PSP in general, and software development in general. *Conclusion validity* is concerned with the relationship between treatment and outcome, and the ability to draw conclusions. *Construct validity* is about generalizing the result to the theory behind the experiment.

The internal validity within the course is probably not a problem. The large number of tests (equal to the number of students) ensures that internal validity is good.

Concerning the external threats, it is highly probable that similar results would be obtained when running the course in a similar way at Lund University. It is more difficult to generalize the results to other students, i.e., students not taking the course. They are probably not as interested in software development and hence they come from a different population. The results from the analysis can probably be generalized to other PSP courses, where it is feasible to compare participants based on their background in terms of computer science or electrical engineering or experience of a particular programming language.

The major threat regarding conclusion validity is the quality of the data collected during the PSP course. The students are expected to deliver a lot of data as part of their work with the course. Thus, there is a risk that the data is faked or simply not correct due to mistakes. The data inconsistencies are, however, not believed to be particularly related to any specific background, and hence the problem is likely the same independent of the background of the individuals. The conclusion validity is hence not considered to be critical.

Construct validity includes two major threats. The first threat is that the measurements as defined may not be appropriate measures of the entities. For example, is "LOC/Development time" a good measure of productivity? The second major threat to construct validity is that it is part of a course where the students are graded. This implies that the students may bias their data, as they believe that it will give them a better grade. However, at the beginning of the course it was emphasized that the grade did not depend on the actual data. The grade was based on timely and proper delivery, and the understanding expressed in the reports handed in during the course.

The results are found for the PSP, but they are likely to hold for software development in general. There is no reason that people coming from different study programs or having different background experience from a particular programming language perform differently between the PSP and software development in general. This is probably valid when talking about differences in background, although the actual size of the difference may vary. The important issue is that there is a difference, and the actual size of the difference is of minor importance.

13.3 Operation

13.3.1 Preparation

The subjects (students) were not aware of what aspects were going to be studied. They were informed that the researchers wanted to study the outcome of the PSP course in comparison with the background of the participants. They were, however, not aware of the actual hypotheses stated. The students, from their point of view, did not primarily participate in an experiment; they were taking a course. All students were guaranteed anonymity.

The survey material was prepared in advance. Most of the other material was, however, provided through the PSP book [113].

13.3.2 Execution

The experiment was executed over 14 weeks, during which the 10 programming assignments were handed in regularly. The data was primarily collected through forms. Interviews were used at the end of the course, primarily to evaluate the course and the PSP as such.

The experiment was, as stated earlier, run within a PSP course and in a university environment. The experiment has not been allowed to affect the course objectives. The main difference between running the PSP solely as a course has been the initial survey of the students' background.

13.3.3 Data Validation

Data was collected for 65 students. After the course, the achievements of the students were discussed among the people involved in the course. Data from six students was removed, due to the fact that the data was regarded as invalid or at least questionable. Students have not been removed (at this stage) from the evaluation based on the actual figures, but due to our trust in the delivered data and whether or not the data is believed to be representative. The six students were removed due to the following:

- Data from two students was not filled in properly.
- One student finished the course much later than the rest, and that student had a long period when no work was done with the PSP. This may have affected the data.
- Data from two students was removed based on the fact that they delivered their assignments late and required considerably more support than the other students did; hence it was judged that the extra advice may have affected their data.

- Finally, one student was removed based on the fact that their background was completely different than that of the others.

This means removing six students out of the 65, hence leaving 59 students for statistical analysis and interpretation of the results.

13.4 Analysis and Interpretation

13.4.1 Descriptive Statistics

As a first sub-step in analyzing the data, descriptive statistics are used to visualize the data collected.

Study Program vs. Productivity Figure 13.1 shows the productivity for the two study programs, when dividing the population into classes based on productivity. The first class includes those with a productivity between 5 and 10 lines of code per hour. Thus, the eighth class includes those with a productivity between 40 and 45 lines of code per hour. From Fig. 13.1, it is possible to see that students from the Electrical Engineering program (EE) seem to have a lower productivity. Moreover, it is notable that the variation of the distribution seems to be larger among the students from the Computer Science and Engineering program (CSE). In total, there are 32 CSE students and 27 EE students. The mean value for CSE students is 23.0 with a standard deviation of 8.7, and for the EE students the mean value is 16.4 with a standard deviation of 6.3. To gain an even better understanding of the data, a box plot is drawn (see Fig. 13.2).

The whiskers in the box plot are constructed as proposed by Frigge et al. [81] and discussed in Chap. 11. For the whiskers, a value that is the length of the

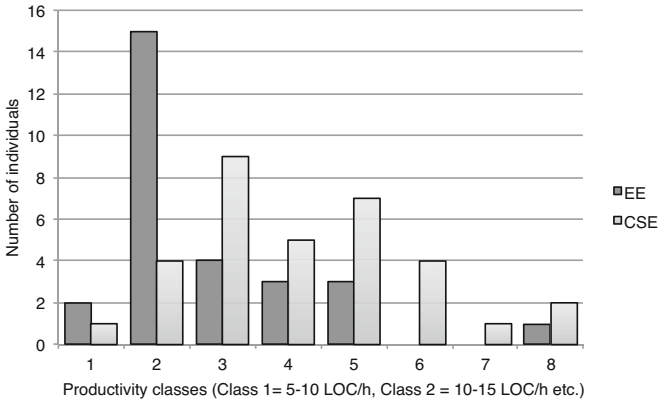


Fig. 13.1 Frequency distribution for the productivity (in classes)

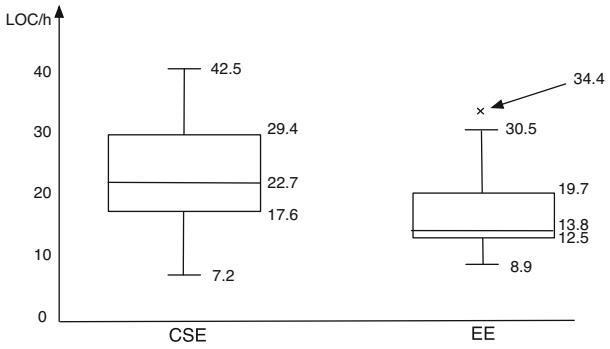


Fig. 13.2 Box plot of productivity for the two study programs

Table 13.1 Faults/KLOC for the different C experience classes

Class[a]	Number of students	Median value of faults/KLOC	Mean value of faults/KLOC	Standard deviation of faults/KLOC
1	32	66.8	82.9	64.2
2	19	69.7	68.0	22.9
3	6	63.6	67.6	20.6
4	2	63	63.0	17.3

[a] The different experience classes are explained in Sect. 13.2.2

box multiplied by 1.5 is added or subtracted from the upper and lower quartiles respectively. For example, for the CSE students (see Fig. 13.2): median = 22.7, box length = $29.4 - 17.6 = 11.8$, the upper tail becomes: $29.4 + 1.5 * 11.8 = 47.1$. There is, however, an exception to this rule, namely that the upper and lower tails should never be higher or lower than the highest and lowest values in the data set; hence the upper tail becomes 42.5, which is the highest value. This exception is introduced to avoid negative values or other types of unrealistic values. The other values in Fig. 13.2 are found in a similar way.

From Fig. 13.2, it can be seen that there is a clear pattern that the EE students have a lower productivity. Thus, it may be possible to identify the difference statistically in a hypothesis test. The t-test is used below.

It is also important to look at outliers in comparison to the upper and lower tails. For the CSE students, there are no values outside the tails. For the EE students, there is one value outside the tails, i.e., 34.4. This value is not considered an outlier since it is not judged as an extreme value. It is an unusual value, but it is determined to keep the value in the analysis.

C experience vs. faults/KLOC The number of students for each class of C experience is shown in Table 13.1, together with the mean and median values, and standard deviation for respective class.

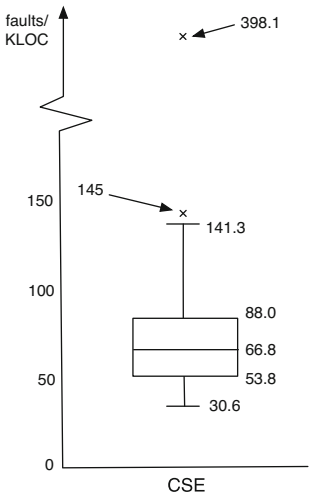


Fig. 13.3 Box plot for faults/KLOC for class 1

From Table 13.1, it can be seen that the distribution is skewed toward no or limited experience of C. If we look at the mean values of faults/KLOC, there seems to be a tendency for the more experienced students to create fewer faults. The standard deviation is, however, extremely large, and the median varies unexpectedly in comparison with the mean value and the underlying hypothesis. The standard deviation for the first class is very high and a further investigation of the data is recommended. Thus, box plot can be used for this data set too.

Box plots are constructed for all four experience classes. The plots for classes 2, 3, and 4 reveal nothing; all values are within the boundaries of the whiskers, and hence the upper and lower tails become equal to the highest and lowest value respectively. The box plot for the first class is more interesting, and it is shown in Fig. 13.3.

From Fig. 13.3, it can be seen that the lower tail is equal to the lowest value of faults/KLOC. The upper tail on the other hand is not equal to the highest value, and hence there are one or more unusual values. There are actually two unusual values, namely 145 and 398.1. The latter value is an extreme one; it is more than 10 times higher than the lowest value. It is also almost three times as high as the second highest value. Thus, it is possible to conclude that the high standard deviation can be explained with the extreme value. For the second hypothesis, the ANOVA test is used.

The descriptive statistics have provided a better insight into the data, both in terms of what can be expected from the hypothesis testing and potential problems caused by outliers.

13.4.2 Data Reduction

Data reduction can always be debated, since as soon as data points are removed information is lost. Two separate ways of reducing data can be identified:

- Single data points can be removed, for example, outliers.
- The data can be analyzed and based on the analysis it may be concluded that due to high inter-correlation between some variables, some measures should be combined into some more abstract measure.

This means that it is possible to either remove data points or reduce the number of variables. In the case of removing data points, the main candidates are the outliers. It is by no means obvious that all outliers should be removed, but they are certainly candidates for removal. It is important to remember that data points should not just be removed because they do not fit with the belief or hypothesis. On the other hand, it is important to remove data points which may make a completely valid relationship invalid, due to the fact that, for example, an extreme outlier is included, which is not expected if replicating the study.

Statistical methods for data reduction are needed to reduce the number of variables. Some examples are principal component analysis and factor analysis [124, 125, 165]. These types of methods are not considered here, as the objective is not to reduce the number of variables.

It is probably better to be restrictive in reducing a data set, as there is always a risk that we aim for a certain result. Thus, for the data presented above, it was decided to only remove the extreme outlier for the number of faults/KLOC. After removing the extreme outlier, the data for class 1 is summarized in Table 13.2.

The removal of the outlier decreased the mean value and standard deviation considerably. The mean number of faults/KLOC is still highest for class 1. However, the differences between the classes are not that large. After reducing the second data set with one data point, it is not possible to perform the statistical test. This is where the hypotheses are evaluated.

13.4.3 Hypothesis Testing

The first hypothesis regarding higher productivity for students following the Computer Science and Engineering program is evaluated using a t-test. An ANOVA test

Table 13.2 Faults/KLOC for the different C experience class 1

Class	Number of students	Median value of faults/KLOC	Mean value of faults/KLOC	Standard deviation of faults/KLOC
1	31	66	72.7	29.0

Table 13.3 Results from the t-test

Factor	Mean diff.	Degrees of freedom (DF)	t-value	p-value
CSE vs. EE	6.1617	57	3.283	0.0018

Table 13.4 Results from the ANOVA test

Factor: C vs. Faults/KLOC	Degrees of freedom (DF)	Sum of squares	Mean square	F-value	p-value
Between treatments	3	3483	1160.9	0.442	0.7236
Error	55	144304	2623.7		

is applied to evaluate the hypothesis that more experience in C means fewer faults/KLOC.

Study Program vs. Productivity The results from the t-test (unpaired, two-tailed) are shown in Table 13.3.

From Table 13.3, it can be concluded that H_0 is rejected. There is a significant difference in productivity for students coming from different study programs. The p-value is very low so the results are highly significant. The actual reason for the difference has to be further evaluated.

C Experience vs. Faults/KLOC This hypothesis is evaluated with an ANOVA test (factorial). The results of the analysis are shown in Table 13.4.

The results from the analysis are not significant, although some differences are observed in terms of mean value (see above) it is not possible to show that there is a significant difference in terms of number of faults/KLOC based on C experience.

Since the numbers of students in classes 3 and 4 are very limited, classes 2, 3, and 4 are grouped together to study the difference between class 1 and the rest. A t-test was performed to evaluate if it was possible to differentiate between class 1 and the grouping of classes 2–4 into one class. No significant results were obtained.

13.5 Summary

We have investigated two hypotheses:

1. Study program vs. productivity
2. C experience vs. faults/KLOC

We are able to show that students from the Computer Science and Engineering program are more productive. This is in accordance with the expectation, although not formally stated in the hypothesis. The expectation was based on the knowledge

that most students from CSE program have taken more computer science and software engineering courses than those from the Electrical Engineering program.

It is not possible to show with any statistical significance that experience in C influences the number of faults/KLOC. This is interesting in relation to Humphrey's [113] recommendation that one should follow the PSP course with a well-known language to focus on the PSP and not the programming language as such. The results obtained may indicate one or several of the following results:

- The difference will become significant with more students.
- The number of faults introduced is not significantly affected by the prior experience. There may be a tendency to make a certain number of faults when developing software. The type of faults may vary, but the total number of faults introduced is about the same.
- The inexperienced students may write larger programs, which directly affects the number of faults/KLOC.

Other explanations can probably also be found, but they all have one thing in common, which is the need for replication. Thus, replication is an important issue to enable us to understand, and hence control and improve, the way software is developed. Furthermore, other factors must be studied as well.

From a validity point of view, it is reasonable to believe that students (in general) from a computer science program have higher productivity than do students coming from other disciplines. This is more or less inherent from the educational background and it is no surprise.

Since it was not possible to show any statistically significant relation between experience in a programming language and the number of faults/KLOC, there are no conclusions to generalize. Further studies are needed, either through replication of the PSP experiment or similar studies in other environments.

13.6 Conclusion

The presented study is a quasi-experiment, since it compares factors which are not randomly assigned to subjects, but rather inherent properties of the subjects (i.e., educational background). It is conducted with students as subjects, which provides good internal validity at the expense of external validity of the results. Being conducted in the PSP context would help in replication of the study, as the context is very well defined.

The study was conducted over several weeks, which usually would have been a threat to the construct validity. However, since this is a quasi-experiment, this is less of a threat. There is no chance of cheating with the educational background. The students were informed about the future use of their collected data, but no explicit consent was obtained, as it should have. However, at the time of the experiment, ethical concerns were not discussed as much. This has changed for the better over time.

In the analysis, six data points were removed, since they did not consistently follow the experimentation process. Another three data point outside the tails of the box plots were analyzed for being considered as outliers. Only an extreme value was removed, since it would have impacted highly on the standard deviation, and thus ruled the analysis outcome.

Chapter 14
Are the Perspectives Really Different?: Further Experimentation on Scenario-Based Reading of Requirements

Background This chapter presents an experimental study as it was published, with the objective to show an example paper from an international journal. Furthermore, the intention is that it should work as a suitable study to practice reviewing skills on. It is important to notice that the paper has been reviewed and revised based on the feedback before being published in the Empirical Software Engineering journal. This means that the quality is higher than the average submitted experimental paper, although paper standards also have raised over time since its original publication. Reviewing scientific papers is further elaborated in Appendix A.2.

Abstract Perspective-Based Reading (PBR) is a scenario-based inspection technique where several reviewers read a document from different perspectives (e.g. user, designer, tester). The reading is made according to a special scenario, specific for each perspective. The basic assumption behind PBR is that the perspectives find different defects and a combination of several perspectives detects more defects compared to the same amount of reading with a single perspective. This paper presents a study which analyses the differences in the perspectives. The study is a partial replication of previous studies. It is conducted in an academic environment using graduate students as subjects. Each perspective applies a specific modelling technique: use case modelling for the user perspective, equivalence partitioning for the tester perspective and structured analysis for the design perspective. A total of 30 subjects were divided into 3 groups, giving 10 subjects per perspective. The analysis results show that (1) there is no significant difference among the three perspectives in terms of defect detection rate and number of defects found per hour, (2) there is no significant difference in the defect coverage of the three perspectives, and (3) a simulation study shows that 30 subjects is enough to detect relatively small

This chapter was originally published in Empirical Software Engineering: An International Journal, Vol. 5, No. 4, pp. 331–356, and it is included in the book with the permission of Springer.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3_14

perspective differences with the chosen statistical test. The results suggest that a combination of multiple perspectives may not give higher coverage of the defects compared to single-perspective reading, but further studies are needed to increase the understanding of perspective difference.

14.1 Introduction

The validation of requirements documents is often done manually, as requirements documents normally include informal representations of what is required of an intended software system. A commonly used technique for manual validation of software documents is inspections, proposed by Fagan [71]. Inspections can be carried out in different ways and used throughout the software development process for (1) understanding, (2) finding defects, and (3) as a basis for making decisions. Inspections are used to find defects early in the development process, and have shown to be cost effective (e.g. by Doolan [60]).

A central part of the inspection process is the *defect detection* carried out by an individual reviewer reading the document and recording defects (a part of preparation, see Humphrey [112]). Three common techniques for defect detection are Ad Hoc, Checklist and Scenario-based reading [200]. Ad Hoc detection denotes an unstructured technique which provides no guidance, implying that reviewers detect defects based on their personal knowledge and experience. The checklist detection technique provides a list of issues and questions, capturing the knowledge of previous inspections, helping the reviewers to focus their reading. In the scenario-based approach, different reviewers have different responsibilities and are guided in their reading by specific scenarios which aim at constructing a model, instead of just passive reading.

A scenario[1] here denotes a script or procedure that the reviewer should follow. Two variants of scenario-based reading have been proposed: Defect-Based Reading [200] and Perspective-Based Reading [25]. The former (subsequently denoted DBR) concentrates on specific defect classes, while the latter (subsequently denoted PBR) focuses on the points of view of the users of a document.

Another part of the inspection process is the *compilation of defects* into a consolidated defect list where all individual reviewers' defect lists are combined. This step may include the removal of false positives (reported defects that were not considered to be actual defects) as well as the detection of new defects. This step is often done in a structured *inspection meeting* where a *team* of reviewers participate.

[1] There is considerable risk for terminology confusion here, as the term *scenario* also is used within requirements engineering to denote a sequence of events involved in an envisaged usage situation of the system under development. A *use case* is often said to cover a set of related (system usage) scenarios. In scenario-based reading, however, the term scenario is a meta-level concept, denoting a procedure that a reader of a document should follow during inspection.

The effectiveness of the team meeting has been questioned and studied empirically by Votta [264] and Johnson and Tjahjono [121].

This paper describes research on scenario-based reading with a PBR approach. The research method is empirical and includes a formal factorial experiment in an academic environment. The presented experiment is a partial replication of previous experiments in the area and focuses on refined hypotheses regarding the differences among the perspectives in PBR. The paper concentrates on defect detection by *individual reviewers*, while the team meeting aspects are not included.

The structure of the paper is as follows. Section 14.2 gives an overview of related work by summarising results from previously conducted experiments in requirements inspections with a scenario-based approach. Section 14.3 includes the problem statement motivating the presented work. In Sect. 14.4, the experiment plan is described including a discussion on threats to the validity of the study, and Sect. 14.5 reports on the operation of the experiment. The results of the analysis is given in Sect. 14.6, and Sect. 14.7 includes an interpretation of the results. Section 14.8 provides a summary and conclusions.

14.2 Related Work

The existing literature on empirical software engineering includes a number of studies related to inspections, where formal experimentation has shown to be a relevant research strategy [274]. The experiment presented in this paper relates to previous experiments on inspections with a scenario-based approach. The findings of a number of experiments on scenario-based inspection of requirements documents are summarised below.

1. The *Maryland-95* study [200] compared DBR with Ad Hoc and Checklist in an academic environment. The experiment was run twice with 24 subjects in each run. The requirements documents used were a water level monitoring system (WLMS, 24 pages) and an automobile cruise control system (CRUISE, 31 pages).

   Result 1: DBR reviewers have significantly higher defect detection rates than either Ad Hoc or Checklist reviewers.
   Result 2: DBR reviewers have significantly higher detection rates for those defects that the scenarios were designed to uncover, while all three methods have similar detection rates for other defects.
   Result 3: Checklist reviewers do *not* have significantly higher detection rates than Ad Hoc reviewers.
   Result 4: Collection meetings produce *no* net improvement in the detection rate—meeting gains are offset by meeting losses.

2. The *NASA* study [25] compared PBR with Ad Hoc in an industrial environment. The experiment consisted of a pilot study with 12 subjects and a second main

run with 13 subjects. There were two groups of requirements documents used; general requirements documents: an automatic teller machine (ATM, 17 pages), a parking garage control system (PG, 16 pages); and two flight dynamics requirements documents (27 pages each).

Result 1: Individuals applying PBR to general documents have significantly higher detection rates compared to Ad Hoc.
Result 2: Individuals applying PBR to NASA-specific documents do not have significantly higher detection rates compared to Ad Hoc.
Result 3: Simulated teams applying PBR to general documents have significantly higher detection rates compared to Ad Hoc.
Result 4: Simulated teams applying PBR to NASA-specific documents have significantly higher detection rates compared to Ad Hoc.
Result 5: Reviewers with more experience do *not* have higher detection rates.

3. The *Kaiserslautern* study [45] compared PBR with Ad Hoc in an academic environment using the ATM and PG documents from the NASA study. The experiment consisted of two runs with 25 and 26 subjects respectively.

Result 1: Individuals applying PBR to general documents have significantly higher detection rates compared to Ad Hoc.
Result 2: Simulated teams applying PBR to general documents have significantly higher detection rates compared to Ad Hoc.
Result 3: The detection rates of five different defect classes are *not* significantly different among the perspectives.

4. The *Bari* study [82] compared DBR with Ad Hoc and Checklist in an academic environment using the WLMS and CRUISE documents from the Maryland-95 study. The experiment had one run with 30 subjects.

Result 1: DBR did *not* have significantly higher defect detection rates than either Ad Hoc or Checklist.
Result 2: DBR reviewers did *not* have significantly higher detection rates for those defects that the scenarios were designed to uncover, while all three methods had similar detection rates for other defects.
Result 3: Checklist reviewers did *not* have significantly higher detection rates than Ad Hoc reviewers.
Result 4: Collection meetings produced *no* net improvement in the detection rate—meeting gains where offset by meeting losses.

5. The *Trondheim* study [239] compared the NASA study version of PBR with a modified version of PBR (below denoted PBR2) where reviewers were given more instructions on how to apply perspective-based reading. The study was conducted in an academical environment using the ATM and PG documents from the NASA study. The experiment consisted of one run with 48 subjects.

Result 1: PBR2 reviewers did *not* have significantly higher defect detection rates than PBR.

Result 2: Individuals applying PBR2 reviewed significantly longer time compared to those who applied PBR.
Result 3: Individuals applying PBR2 suggested significantly fewer potential defects compared to those who applied PBR.
Result 4: Individuals applying PBR2 had significantly lower productivity and efficiency than those who applied PBR.

6. The *Strathclyde* study [176] compared DBR with Checklist in an academic environment using the WLMS and CRUISE documents from the Maryland study. The experiment consisted of one run with 50 subjects.

Result 1: In the WLMS document, DBR did *not* have significantly higher defect detection rates than Checklist.
Result 2: In the CRUISE document, DBR had significantly higher defect detection rates than Checklist.
Result 3: Collection meetings produced *no* net improvement in the detection rate—meeting gains were offset by meeting losses.

7. The *Linköping* study [215] compared DBR with Checklist in an academic environment using the WLMS and CRUISE documents from the Maryland study. More defects were added to the list of total defects. The experiment consisted of one run with 24 subjects.

Result 1: In the WLMS document, DBR reviewers did *not* have significantly higher defect detection rates than Checklist reviewers.
Result 2: In the CRUISE document, DBR reviewers did *not* have significantly higher detection rates than Checklist reviewers.

8. The *Maryland-98* study [223] compared PBR with Ad Hoc in an academic environment using the ATM and PG documents from the Maryland study. The experiment consisted of one run with 66 subjects.

Result 1: PBR reviewers had significantly higher defect detection rates than Ad Hoc reviewers.
Result 2: Individuals with high experience applying PBR did *not* have significantly[2] higher defect detection rates compared to Ad Hoc.
Result 3: Individuals with medium experience applying PBR had significantly higher defect detection rates compared to Ad Hoc.
Result 4: Individuals with low experience applying PBR had significantly higher defect detection rates compared to Ad Hoc.
Result 5: Individuals applying PBR had significantly lower productivity compared to those who applied Ad Hoc.

9. The *Lucent* study [199] replicated the Maryland-95 study in an industrial environment using 18 professional developers at Lucent Technologies. The

[2] Results 2–4 of the Maryland-98 study apply a significance level of 0.10, while 0.05 is the chosen significance level in all other results.

Table 14.1 Summary of studies

Study	Purpose	Environment	Subjects	Significant?
Maryland-95	DBR vs. AdHoc and Checklist	Academic	24+24	YES
Bari	DBR vs. AdHoc and Checklist	Academic	30	NO
Strathclyde	DBR vs. Checklist	Academic	50	Inconclusive
Linköping	DBR vs. Checklist	Academic	24	NO
Lucent	DBR vs. AdHoc and Checklist	Industrial	18	YES
NASA	PBR vs. AdHoc	Industrial	12+13	YES
Kaiserslautern	PBR vs. AdHoc	Academic	25+26	YES
Trondheim	PBR vs. PBR2	Academic	48	NO
Maryland-98	PBR vs. AdHoc	Academic	66	YES

replication was successful and completely corroborated the results from the Maryland-95 study.

The results of the different studies vary substantially. An attempt to systematically address the combined knowledge, gained from experiments and replications is reported by Hayes [104], where meta-analysis is applied to the results of the Maryland-95, Bari, Strathclyde, Linköping and Lucent studies. It is concluded from the meta-analysis that the effect sizes for the inspection methods are inhomogeneous across the experiments. The Maryland-95 and Lucent studies show most similar results, and an interpretation of the meta-analysis identifies characteristics which make them different from the other three studies: (1) they are conducted in a context where the subjects are more familiar with the notation used, (2) they are conducted in the US where cruise control are more common in cars than in Europe where the other three studies are performed. These hypotheses are, however, not possible to test with the given data, and thus more experimentation is needed.

Table 14.1 includes a summary of the presented studies. The Maryland-95, NASA, Kaiserslautern, Maryland-98, and Lucent studies indicate that a scenario-based approach gives higher detection rate. The Bari, Strathclyde, and Linköping studies could, however, not corroborate these results, which motivates further studies to increase the understanding of scenario-based reading.

Many of the studies concluded that real team meetings were ineffective in terms of defect detection. (There may of course be other good reasons for conducting team meetings apart from defect detection, such as consensus building, competence sharing, and decision making.)

The study presented here is subsequently denoted the *Lund* study. The Lund study is a partial replication of the NASA study, and is based on a lab package [26] provided by the University of Maryland in order to support empirical investigations of scenario-based reading. The problem statement motivating the Lund study is given in the subsequent section.

14.3 Research Questions

The previous studies, summarised in Sect. 14.2, have mainly concentrated on comparing scenario-based reading with checklist and Ad Hoc techniques in terms of defect detection rates. The objective of the Lund study is, however, to investigate the basic assumption behind scenario-based reading, that the different perspectives find different defects. Another interest is the efficiency of the different perspectives in terms of defects detected per hour. The following two questions are addressed:

1. Do the perspectives detect different defects?
2. Is one perspective superior to another?

There are two aspects of superiority that are addressed: *effectiveness*, i.e. how high fraction of the existing defects are found (detection rate), and *efficiency*, i.e. how many defects are found per time unit.

The perspectives proposed by Basili et al. [25] are designer, tester and user. The users are important stakeholders in the software development process, and especially when the requirements are elicited, analysed and documented. The user role in PBR is focused on detecting defects at a high abstraction level related to system usage, while the designer is focused on internal structures and the tester is focused on verification.

Previous studies have mainly concentrated on the effectiveness in terms of detection rate. From a software engineering viewpoint it is important also to assess the efficiency (e.g. in terms of detected defects per time unit), as this factor is important for a practitioner's decision to introduce a new reading technique. The specific project and application domain constraints then can, together with estimations of how much effort is needed, be a basis for a trade-off between quality and cost.

One main purpose of PBR is that the perspectives detect different kinds of defects in order to minimise the overlap among the reviewers. Hence, a natural question is whether reviewers do find different defects or not. If they detect the same defects, the overlap is not minimised and PBR does not work as it was meant to. If all perspectives find the same kinds of defects it may be a result of (1) that the scenario-based reading approach is inappropriate, (2) that the perspectives may be insufficiently supported by their accompanying scenarios, or (3) that other perspectives are needed to gain a greater coverage difference. The optimal solution is to use perspectives with no overlap and as high defect detection rate as possible, making PBR highly dependable and effective. The Lund study addresses the overlap by investigating whether the perspectives detect different defects.

Research question 1 is also interesting from a defect content estimation perspective. The capture-recapture approach to defect content estimation uses the overlap among the defects that the reviewers find to estimate the number of remaining defects in a software artefact [66, 172]. The robustness of capture-recapture using PBR is studied by Thelin and Runeson [249], with the aim of investigating capture-recapture estimators applied to PBR inspections under the hypothesis that PBR

works according to its underlying assumption. In the Lund study it is investigated whether the assumptions of PBR are factual. Hence, the Lund study and the Thelin and Runeson [249] study complement each other in order to answer the question whether capture-recapture estimations can be used for PBR inspections.

14.4 Experiment Planning

This section describes the planning of the reading experiment. The planning includes the definition of dependent and independent variables, hypotheses to be tested in the experiment, experiment design, instrumentation and an analysis of threats to the validity of the experiment [274].

The reading experiment is conducted in an academical environment with close relations to industry. The subjects are fourth-year students at the Master's programmes in Computer Science & Engineering and Electrical Engineering at Lund University.

14.4.1 Variables

The independent variables determine the cases for which the dependent variables are sampled. The purpose is to investigate different reading perspectives and methods, applied to two objects (requirements documents). The inspection objects are the same as in the University of Maryland lab package [26], and the design and instrumentation are also based on this lab package. The variables in the study are summarised in Table 14.2 together with brief explanations.

14.4.2 Hypotheses

Perspective-Based Reading is assumed to provide more efficient inspections, as different reviewers take different perspectives making the defect overlap smaller [25]. The objective of the study is to empirically test whether these assumptions are true. In consequence, hypotheses related to performance of different perspectives are stated below. The three null hypotheses address efficiency, effectiveness and distribution over perspectives.

- $H_{0,EFF}$. The perspectives are assumed to have the same finding efficiency, i.e. the number of defects found per hour of inspection is not different for the various perspectives.

Table 14.2 Variables

	Name	Values	Description
Independent Variables	PERSP	{U,T,D}	One of three perspectives is applied by each subject: User, Tester, and Designer
	DOC	{ATM,PG}	The inspection objects are two requirements documents: one for an automatic teller machine (ATM) and one for a parking garage control system (PG). The ATM document is 17 pages and contains 29 defects. The PG document is 16 pages and contains 30 defects
Controlled Variable	EXPERIENCE	Ordinal	The experience with user, tester, design perspectives is measured on a five-level ordinal scale and used in the allocation of subjects to perspectives. (See Sects. 14.4.3 and 14.6.4)
Dependent Variables	TIME	Integer	The time spent by each reviewer in individual preparation is recorded by all subjects. The time unit used is minutes
	DEF	Integer	The number of defects found by each reviewer is recorded, excluding false positives. The false positives are removed by the experimenters, in order to ensure that all defect candidates are treated equally
	EFF	60*DEF/TIME	The defect finding efficiency, i.e. the number of defects found per hour, is calculated as (DEF*60)/TIME
	RATE	DEF/TOT	The defect finding effectiveness, i.e. the fraction of found defects by total number of defects (also called detection rate) is calculated as DEF divided by the total number of known defects contained in the inspected documents
	FOUND	Integer	The number of reviewers belonging to a certain perspective, which have found a certain defect in a specific document is recorded. This variable is used for analysing defect finding distributions for different perspectives

- $H_{0,RATE}$. The perspectives are assumed to have the same effectiveness or detection rates, i.e. the fraction of defects identified is not different for the various perspectives.
- $H_{0,FOUND}$. The perspectives are assumed to find the same defects, i.e. the distributions over defects found are the same for the different perspectives.

Table 14.3 Experiment design

		PERSP		
		User	**Designer**	**Tester**
DOC	ATM	5	5	5
	PG	5	5	5

14.4.3 Design

To test these hypotheses an experiment with a factorial design [180] is used with two factors (PERSP and DOC). The design is summarised in Table 14.3. The experiment varies the three perspectives over two documents.

The assignment of an individual subject to one of the three PBR perspectives (U, D, T), was conducted based on their reported experience (see Sect. 14.6.4), similar to the NASA study [25]. The objective of experience-based perspective assignment is to ensure that each perspective gets a fair distribution of experienced subjects, so that the outcome of the experience is affected by perspective difference rather than experience difference. The experience questionnaire required the subjects to grade their experience with each perspective on a five level ordinal scale. The subjects were then sorted three times, giving a sorted list of subjects for each perspective with the most experienced first. Within the same experience level, the subjects were placed in random order. The subjects were then assigned to perspectives by selecting a subject on top of a perspective list and removing this subject in the other lists before continuing with the next perspective in a round robin fashion starting with a randomly selected perspective, until all subjects were assigned a perspective.

The instruments of the reading experiment consist of two requirements documents and reporting templates for time and defects. These instruments are taken from the University of Maryland lab package [26] and are reused with minimal changes.

The factorial design described above is analysed with descriptive statistics (bar plots and box plots) and analysis of variance (ANOVA) [180] for the hypotheses $H_{0,EFF}$, and $H_{0,RATE}$. For the $H_{0,FOUND}$ hypothesis a Chi-square test [228] is used together with a correlation analysis [207].

14.4.4 Threats to Validity

The validity of the results achieved in experiments depends on factors in the experiment settings. Different types of validity can be prioritised depending on the goal of the experiment. In this case, threats to four types of validity are analysed [50, 274]: conclusion validity, internal validity, construct validity and external validity.

Conclusion validity concerns the statistical analysis of results and the composition of subjects. In this experiment, well known statistical techniques are applied

which are robust to violations of their assumptions. One general threat to conclusion validity is, however, the low number of samples, which may reduce the ability to reveal patterns in the data. In particular, there are few samples for the Chi-square test, which is further elaborated in Sect. 14.6.3.

Internal validity concerns matters that may affect the independent variable with respect to causality, without the researchers knowledge. There are two threats to internal validity in this experiment, selection and instrumentation. The experiment was a mandatory part of a software engineering course, thus the selection of subjects is not random, which involves a threat to the validity of the experiment. The requirements documents used may also affect the results. The documents are rather defect-prone and additional issues in the documents could be considered as defects. On the other hand, it is preferable to have the same definition of defects as in the previous studies for comparison reasons. Other threats to internal validity are considered small. Each subject was only allocated to a single object and a single treatment, hence there is no threat of maturation in the experiment. The subjects applied different perspectives during inspection, but the difference among perspectives are not large enough to suspect compensatory equalisation of treatments or compensatory rivalry. The subjects were also told that their grading in the course was not depending on their performance in the experiment, only on their serious attendance. There is of course a risk that the subjects lack motivation; they may, for example, consider their participation a waste of time or they may not be motivated to learn the techniques. The teacher in the course in which the experiment was performed has, however, made a strong effort in motivating the students. It was clearly stated that a serious participation was mandatory for passing the course. It is the teacher's opinion that the students made a very serious attempt in their inspection.

Construct validity concerns generalisation of the experiment result to concept or theory behind the experiment. A major threat to the construct validity is that the chosen perspectives or the reading techniques for the perspectives may not be representative or good for scenario-based reading. This limits the scope for the conclusions made to these particular perspectives and techniques. Other threats to the construct validity are considered small. The subjects did not know which hypotheses were stated, and were not involved in any discussion on advantages and disadvantages of PBR, thus they were not able to guess what the expected results were.

External validity concerns generalisation of the experiment result to other environments than the one in which the study is conducted. The largest threat to the external validity is the use of students as subjects. However, this threat is reduced by using fourth-year students which are close to finalise their education and start working in industry. The setting is intended to resemble a real inspection situation, but the process that the subjects participate in is not part of a real software development project. The assignments are also intended to be realistic, but the documents are rather short, and real software requirements documents may include many more pages. The threats to external validity regarding the settings and

assignments are, however, considered limited, as both the inspection process and the documents resemble real cases to a reasonable extent.

It can be concluded that there are threats to the construct, internal and external validity. However, these are almost the same as in the original studies. Hence, as long as the conclusions from the experiment are not drawn outside the limitations of these threats, the results are valid.

14.5 Experiment Operation

The experiment was run during spring 1998. The students were all given a two hour introductory lecture where an overview of the study was given together with a description of the defect classification. A questionnaire on experience was given and each subject was assigned to a perspective, as described in Sect. 14.4.3. The students were informed that the experiment was a compulsory part of the course, but the grading was only based on serious participation in the study and not on the individual performance of the students. The anonymity of the students was guaranteed.

A two hour exercise was held, where the three PBR perspectives were described and illustrated using a requirements document for a video rental system (VRS). During the second hour of the exercise, the subjects were practising their own perspective reading technique for the VRS document, and had the opportunity to ask questions. The data collection forms were also explained and used during the exercise. The perspective-based reading of the VRS document was completed by the students on their own after the classroom hours.

The hand-outs for the experiment, which were handed out during the exercise, included the following instrumentation tools:

1. Defect Classification which describes defect classes to be used in the defect list.
2. Time Recording Log for recording the time spent on reading.
3. Defect List for recording the found defects.
4. Reading Instruction, specific for the user, designer, and tester perspectives respectively.
5. Modelling Forms, specific for the user, designer, and tester perspectives respectively.
6. The requirements document (either ATM or PG).

The students were instructed not to discuss the ATM or PG documents and the defects that they find. They were allowed to discuss the PBR perspectives in relation to the VRS document before they started with the actual data collection.

14.6 Data Analysis

This section presents the statistical analysis of the gathered data. The data were collected from the hand-ins from subjects. Each defect in each subject's defect log was compared with the original "correct" defect list provided by the University of Maryland lab package. In a meeting, the authors discussed each defect and decided whether it corresponded to a "correct" defect. If no corresponding "correct" defect was found, the reported defect was considered a false positive.[3] The reported time spent was also collected and the EFF, RATE, and FOUND measures were calculated. The data sets are provided in Sects. 14.9 and 14.10 in Tables 14.6, 14.7, and 14.8.

14.6.1 Individual Performance for Different Perspectives

Box-plots[4] of individual performance in terms of number of defects found per hour (EFF), and the fraction of found defects against the total number of defects (RATE), are shown in Fig. 14.1. The box-plots are split by document and perspective.

For EFF, the Tester perspective on the PG document has a higher mean than the User and Designer perspectives, while for the ATM document, the Designer perspective has a higher mean. For RATE the Designer means are higher compared to the User and Tester perspectives for both documents. There are, however, too few data points per group for any further interpretation of the box-plots, with respect to outliers and skewness.

When several dependent variables are measured, the multi-variate analysis of variance (MANOVA) can be used to assess if there exists any statistically significant difference in the total set of means. The results of MANOVA tests regarding the effect of PERSP reveal no significance and indicate absence of interaction effects. Furthermore, there are no significant differences in the means of EFF, RATE for the PERSP variable, as shown by the analysis of variance (ANOVA) in Tables 14.4 and 14.5. From this analysis it can be concluded that the null hypotheses for EFF and RATE can not be rejected for any of the three perspectives.

[3] Some of the defects that were decided to be false positives may in fact be true defects if the defect list from the Maryland lab package is incomplete. It was decided, however, that it is important from a replication viewpoint that the same list of "correct" defects was used. This decision is not considered to have any significant impact on the result as there were only few false positives that were questionable.

[4] The box-plots are drawn with the box height corresponding to the 25th and 75th percentile, with the 50th percentile (the median) marked in the box. The whiskers correspond to the 10th and 90th percentile.

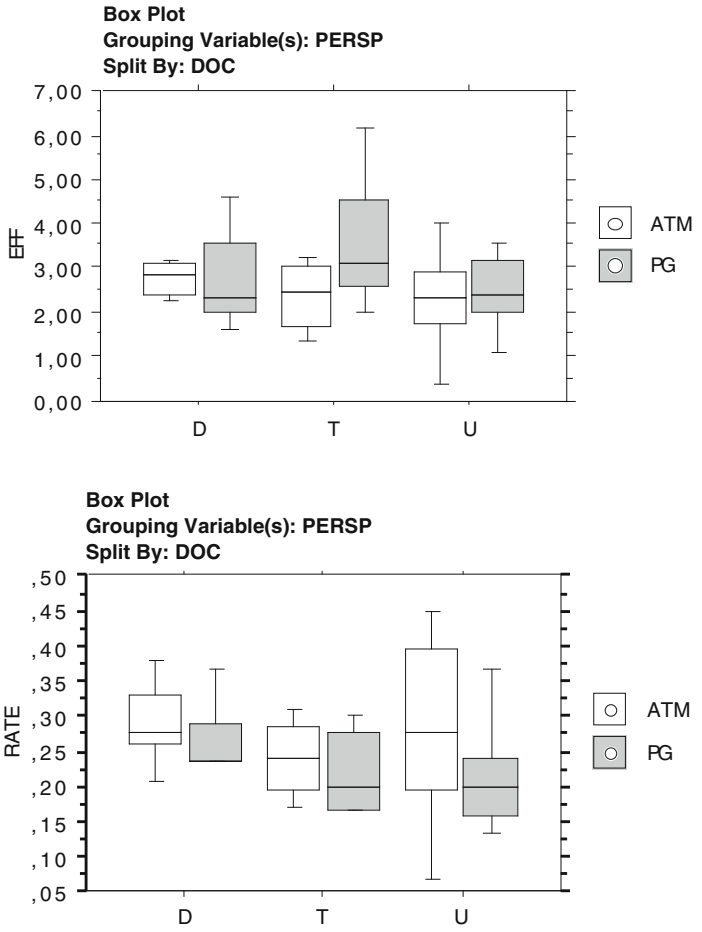


Fig. 14.1 Box plots for EFF and RATE split by DOC and PERSP

Table 14.4 ANOVA table for EFF

	DF	Sum of Sq	Mean Sq	F-Value	p-value	Lambda	Power
PERSP	2	1.751	.875	.737	.4893	1.473	.156
DOC	1	1.640	1.640	1.380	.2516	1.380	.193
PERSP * DOC	2	2.229	1.114	.937	.4055	1.875	.187
Residual	24	28.527	1.189				

14.6.2 Defects Found by Different Perspectives

The hypothesis $H_{0,FOUND}$ regarding the overlap of the found defects among the perspectives, is studied in this section. Descriptive statistics in the form of bar chart

Table 14.5 ANOVA table for RATE

	DF	Sum of Sq	Mean Sq	F-Value	p-value	Lambda	Power
PERSP	2	.012	.006	.802	.4602	1.604	.166
DOC	1	.011	.011	1.488	.2344	1.488	.205
PERSP * DOC	2	.004	.002	.259	.7739	.518	.085
Residual	24	.172	.007				

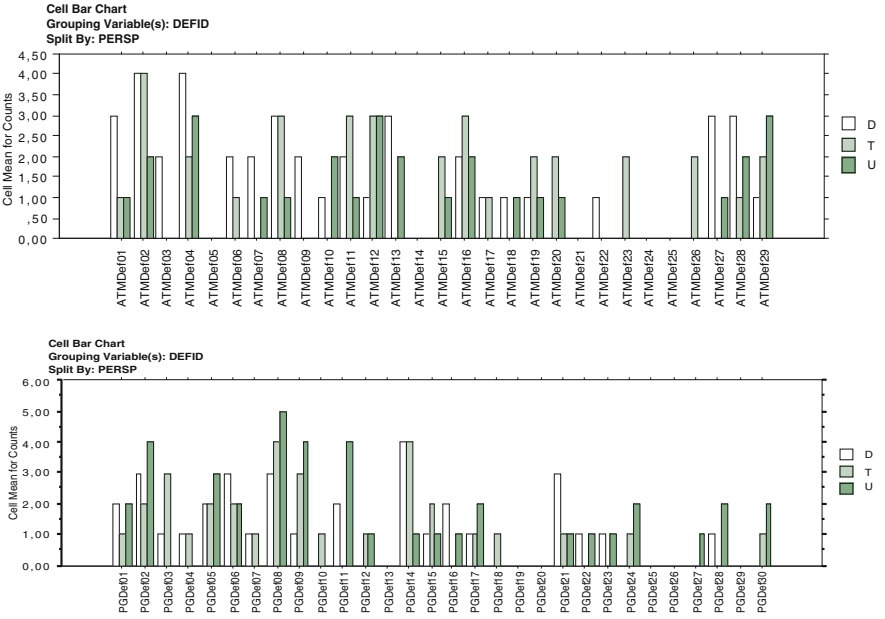


Fig. 14.2 Bar charts illustrating the distribution of number of reviewers that found each defect

plots are shown in Fig. 14.2. For each document the distribution of number of found defects per perspective is shown. There do not seem to be any particular patterns in the different perspective distributions; the defect findings of each perspective seem similarly spread over the defect space. If there had been large differences in the perspective distributions, the bar plot would presumably have groups of defects where one perspective would have a high number of findings while the others would have a low number of findings.

In order to compare the distributions of found defects for each perspective and investigate if there is a significant difference among which defects the perspectives find, a contingency table is created for which a Chi Square test is made [228, pp. 191–194], as shown in Fig. 14.3. The defects that no perspective have found are excluded from the contingency tables (the "Inclusion criteria" in Fig. 14.3), as these cases do not contribute to the testing of differences.

Summary Table for DEFID, PERSP
Inclusion criteria: Counts>0 from PG.data

Num. Missing	0
DF	46
Chi Square	33,951
Chi Square P-Value	,9058
G-Squared	.
G-Squared P-Value	.
Contingency Coef.	,494
Cramer's V	,402

Summary Table for DEFID, PERSP
Inclusion criteria: Counts > 0 from ATM.data

Num. Missing	0
DF	46
Chi Square	41,676
Chi Square P-Value	,6538
G-Squared	.
G-Squared P-Value	.
Contingency Coef.	,535
Cramer's V	,448

Observed Frequencies for DEFID, PERSP
Inclusion criteria: Counts>0 from PG.data

	D	T	U	Totals
PGDef01	2	1	2	5
PGDef02	3	2	4	9
PGDef03	1	3	0	4
PGDef04	1	1	0	2
PGDef05	2	2	3	7
PGDef06	3	2	2	7
PGDef07	1	1	0	2
PGDef08	3	4	5	12
PGDef09	1	3	4	8
PGDef10	0	1	0	1
PGDef11	2	0	4	6
PGDef12	0	1	1	2
PGDef14	4	4	1	9
PGDef15	1	2	1	4
PGDef16	2	0	1	3
PGDef17	1	1	2	4
PGDef18	0	1	0	1
PGDef21	3	1	1	5
PGDef22	1	0	1	2
PGDef23	1	0	1	2
PGDef24	0	1	2	3
PGDef27	0	0	1	1
PGDef28	1	0	2	3
PGDef30	0	1	2	3
Totals	33	32	40	105

Observed Frequencies for DEFID, PERSP
Inclusion criteria: Counts > 0 from ATM.data

	D	T	U	Totals
ATMDef01	3	1	1	5
ATMDef02	4	4	2	10
ATMDef03	2	0	0	2
ATMDef04	4	2	3	9
ATMDef06	2	1	0	3
ATMDef07	2	0	1	3
ATMDef08	3	3	1	7
ATMDef09	2	0	0	2
ATMDef10	1	0	2	3
ATMDef11	2	3	1	6
ATMDef12	1	3	3	7
ATMDef13	3	0	2	5
ATMDef15	0	2	1	3
ATMDef16	2	3	2	7
ATMDef17	1	1	0	2
ATMDef18	1	0	1	2
ATMDef19	1	2	1	4
ATMDef20	0	2	1	3
ATMDef22	1	0	0	1
ATMDef23	0	2	0	2
ATMDef26	0	2	0	2
ATMDef27	3	0	1	4
ATMDef28	3	1	2	6
ATMDef29	1	2	3	6
Totals	42	34	28	104

Fig. 14.3 Chi Square tests and contingency tables for defects found by U,T,D per DOC

The Chi Square P-values are far from significant, indicating that it is not possible with this test and this particular data set to show a difference in the perspectives? defect finding distributions. There are rules of thumb regarding when the Chi Square test can be used [228, pp. 199–200], saying that no more than 20% of the cells should have an expected frequency of less than 5, and no cell should have an expected frequency of less than 1. These rules of thumb are not fulfilled by the data set in this case, but it may be argued that the rules are too conservative and as the expected

ATM Document

Correlation Analysis

	Correlation	P-Value	95% Lower	95% Upper
User, Tester	,480	,0076	,138	,720
User, Designer	,499	,0052	,162	,732
Tester, Designer	,258	,1789	-,120	,570

29 observations were used in this computation.

Correlation Analysis
Inclusion criteria: User > 0 OR Tester > 0 OR Designer > 0 from ATM-ctable.data

	Correlation	P-Value	95% Lower	95% Upper
User, Tester	,357	,0867	-,054	,665
User, Designer	,352	,0915	-,059	,662
Tester, Designer	,043	,8449	-,367	,439

24 observations were used in this computation.

PG Document

Correlation Analysis

	Correlation	P-Value	95% Lower	95% Upper
User, Tester	,463	,0092	,123	,706
User, Designer	,543	,0016	,228	,756
Tester, Designer	,601	,0003	,307	,790

30 observations were used in this computation.

Correlation Analysis
Inclusion criteria: User > 0 OR Tester > 0 OR Designer > 0 from PG-ctable.data

	Correlation	P-Value	95% Lower	95% Upper
User, Tester	,319	,1300	-,097	,640
User, Designer	,414	,0438	,012	,700
Tester, Designer	,493	,0134	,112	,748

24 observations were used in this computation.

Fig. 14.4 Correlation analysis of the perspectives for each document

frequencies in our case are rather evenly distributed, the Chi Square test may still be valid (see further Sect. 14.6.3).

The Chi Square test does not give a measure of the degree of difference. In order to analyse how different (or similar) the perspectives are, a correlation analysis is presented in Fig. 14.4, using the Pearson correlation coefficient [207, pp. 338–340].

Two different correlation analyses are provided for each document, one with all "correct" defects included and one where only those defects are included that were found by at least one reviewer. The latter may be advocated, as we are interested in the differences in the set of defects that are found by each perspective; the defects that no perspective find do not contribute to differences among perspectives.

The P-value indicates if the correlation coefficient is significant, and the confidence intervals presented indicate the range wherein the correlation coefficient is likely to be.

The correlation analysis indicates that there are significantly positive correlations among the perspectives, meaning that when one perspective finds a defect it is likely that others also find it. The only correlation coefficient that is far from significant is the Designer-Tester correlation for the ATM document.

Another way of qualitatively analysing the overlap among the perspectives is Venn-diagrams, as used in the NASA study [25, p. 151].

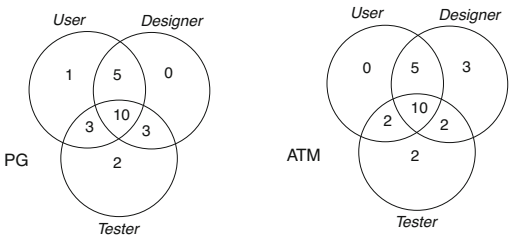


Fig. 14.5 Defect coverage for the PG and ATM documents

For the purpose of comparison we include such diagrams for the Lund study data, as shown in Fig. 14.5. Each defect is categorised in one of seven classes depending on which combinations of perspectives that have a FOUND measure greater than zero. The numbers in the Venn-diagrams indicate how many defects that belong to each class. For example, for the PG document, there are 10 defects which were found by all perspectives, while 5 defects were found by both the user and designer perspectives and only one defect was found solely by the user perspective.

This type of analysis is very sensitive to the number of subjects. It is enough that only one reviewer finds a defect, for the classification to change. The probability that a defect is found increases with the number of reviewers, and if we have a large number of reviewers, the defects will be more likely to be included in the class where all perspectives have found it. This means that this type of analysis is not very robust, and does not provide meaningful interpretations in the general case. In our case, we can at least say that the defect coverage analysis in Fig. 14.5 does not contradict our previous results that we cannot reject the hypothesis that the perspectives are similar with respect the sets of defects that they find. The defects found by all perspectives is by far the largest class.

14.6.3 Is the Sample Size Large Enough?

The outcome of the Lund study is that no significant difference among the perspectives can be detected. A question arises whether this is due to lack of differences in the data, or that the statistical tests are not able to reveal the differences, for example, due to the limited amount of data. In order to evaluate the Chi-square test the perspective defect detection data sets are simulated with stochastic variations among perspectives and the Chi-square test is applied to the simulated data.

The simulation is designed to resemble the experiment presented in the previous section. The difference is that in the simulation case, the probability for detection of a specific defect by a perspective is an independent variable. Furthermore, only the FOUND dependent variable is applied, since the time aspect is not modelled. The simulation model is designed as follows:

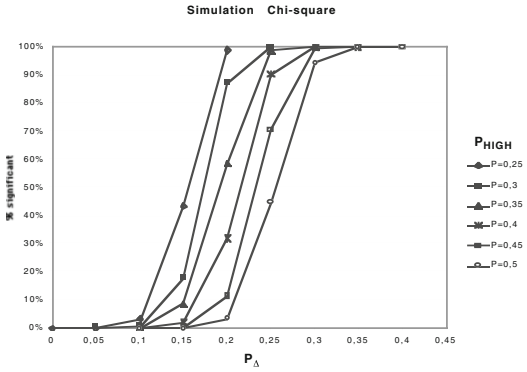


Fig. 14.6 Fraction of significant test results concerning $H_{0,FOUND}$

- The number of defects in each simulated document is 30.
- For every simulated inspection, three perspectives are used with 10 reviewers per perspective. It is assumed that a document contains three different types of defects, which have different probabilities of being detected. One perspective has high probability (P_{HIGH}) to detect one third of the defects and low probability (P_{LOW}) to detect the other two thirds of the defects. The difference between P_{HIGH} and P_{LOW} is denoted P_{Δ}. The probability levels are set to values between 0.05 and 0.5 in steps of 0.05, which are values around the measured mean in the Lund study.
- 1000 runs of each inspection are simulated.

The $H_{0,FOUND}$ hypothesis is tested with the Chi-square test and the results are presented in Fig. 14.6. Each simulated experiment is tested separately. The figure shows the fraction of tests that are rejected for each case. For all simulation cases with P_{Δ} larger than 0.3, the test can significantly show a difference among the simulated perspectives. For simulation cases with P_{HIGH} lower than 0.25, the differences can be shown if P_{Δ} is larger than 0.2. The tests are conducted with a significance level of 0.05. The simulation study shows that differences in FOUND are possible to detect with the Chi-square test, even if the perspective differences are small and the sample size is small.

14.6.4 Experience of Subjects

The experience was measured through a questionnaire which covers each perspective in general, as well as experience with the specific modelling techniques of the three perspectives (use case modelling, equivalence partitioning, and structured analysis). The experience is measured for each general perspective and each specific modelling technique on a five level ordinal scale: 1 = none, 2 = studied in class or from book, 3 = practised in a class project, 4 = used on one project in industry, 5 = used on multiple projects in industry.

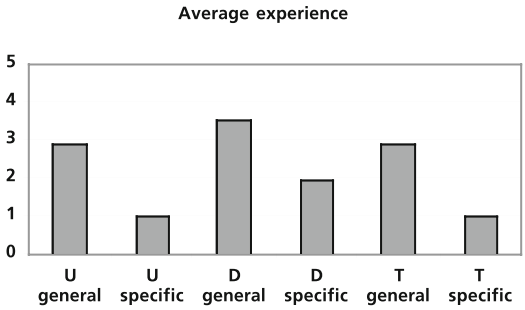


Fig. 14.7 Average experience of subjects regarding their general experience of their perspective and specific experience with their modelling technique

Figure 14.7 shows the average experience for each subject regarding the perspective to which the subject was assigned, both for the perspective in general and for the specific modelling technique.

It can be seen that the allocation of subjects (according to the algorithm explained in Sect. 14.4.3) has, as expected, resulted in a relatively balanced experience profile over the perspectives. It can also be noted that the students had very little industrial experience.

14.7 Interpretations of Results

In this section the data analysis is interpreted with respect to the hypotheses stated in Sect. 14.4.2. The first two hypotheses are tested using ANOVA and the third hypothesis is tested using a Chi-square test. The following three null-hypotheses can not be rejected:

- $H_{0,EFF}$ The perspectives are assumed to find the same number of defects per hour. This hypothesis can *not* be rejected.
- $H_{0,RATE}$ The perspectives are assumed to find the same number of defects. This hypothesis can *not* be rejected.
- $H_{0,FOUND}$ The perspectives are assumed to find the same defects. This hypothesis can *not* be rejected.

It can hence be concluded that there is no significant difference among the three perspectives, user, design and test. This is true for all the three hypotheses, i.e. there is no significant difference in terms of effectiveness or efficiency. Furthermore, there is no significant difference in time spent using the different perspectives, hence, the time spent does not bias in favour of any of the techniques. The lack of difference among the three perspectives does, if the result is possible to replicate and generalise, seriously affect the cornerstones of the PBR. The advantages of PBR are assumed to be that the different perspectives focus on different types of defects, and thus detect different defect sets. This study shows no statistically significant

difference among the sets of defects found by the three perspectives, and thus the advantages of PBR can be questioned.

Threats to the conclusion validity of the results are that the number of samples is low, in particular for the Chi-square test. However, a simulation study reveals that the Chi-square test can with 30 subjects detect differences among perspectives for relatively small differences in detection probability. Furthermore, the bar charts over the defects found by different perspectives (see Fig. 14.2) do not indicate any clear pattern, which supports the non-significant results. The ANOVA statistics are applied within acceptable limits, and these do not show any difference among the perspectives. The specific perspectives and the reading techniques for the perspectives might also be a threat to the validity of the results, when trying to apply the results to scenario-based reading in general.

The validity threat regarding the motivation of subjects can be evaluated by comparing the detection rates of the Lund study with other studies. The individual PBR detection rate for the NASA study [25] was on average 0.249 for the pilot study and 0.321 for the main run, while the Lund study shows an average individual PBR detection rate of 0.252. The rates are comparable, supporting the assumption that the subjects in this study was as motivated as in the NASA study.

Other threats to the validity in Sect. 14.4.4 are not considered differently in the light of the result.

14.8 Summary and Conclusions

The study reported in this paper is focused on the evaluation of Perspective Based Reading (PBR) of requirements documents. The study is a partial replication of previous experiments in an academic environment based on the lab package from University of Maryland [26].

The objective of the presented study is twofold:

1. Investigate the differences in the performance of the perspectives in terms of effectiveness (defect detection rate) and efficiency (number of found defects per hour).
2. Investigate the differences in defect coverage of the different perspectives, and hence evaluate the basic assumptions behind PBR supposing that different perspectives find different defects.

The experiment setting includes two requirements documents and scenarios for three perspectives (*user* applying use case modelling, *designer* applying structured analysis, and *tester* applying equivalence partitioning). A total of 30 MSc students were divided into 3 groups, giving 10 subjects per perspective.

In summary the results from the data analysis show that:

1. There is no significant difference among the user, designer and tester perspectives in terms of defect detection rate and number of defects found per hour.

2. There is no significant difference in the defect coverage of the three perspectives.

The interpretation of these results suggests that a combination of multiple perspectives may not give higher defect coverage compared to reading with only one perspective.

The results contradict the main assumptions behind PBR. Some of the previous studies, summarised in Sect. 14.2, have shown significant advantages with Scenario-based Reading over Ad Hoc inspection, but no statistical analysis on the difference among perspective performance is made in any of the studies reported in Sect. 14.2. Furthermore, the previous studies in Sect. 14.2 have not taken the efficiency into account (number of defects found per hour), but concentrates on detection rate as the main dependent variable. From a software engineering perspective, where the cost and efficiency of a method are of central interest, it is very interesting to study not only the detection rate, but also if a method can perform well within limited effort.

There are a number of threats to the validity of the results, including:

1. The setting may not be realistic.
2. The perspectives may not be optimal.
3. The subjects may not be motivated or trained enough.
4. The number of subjects may be too small.

It can be argued that the threats to validity are under control, based on the following considerations: (1) The inspection objects are similar to industrial requirements documents; (2) The perspectives are motivated from a software engineering process view; (3) The subjects were 4th year students with a special interest in software engineering attending an optional course which they have chosen out of their own interest, and further, many companies have a large fraction of employees with fresh exams; (4) The presented simulation study shows that relatively small differences among the perspectives can be detected with the chosen analysis for the given number of data points.

A single study, like this, is no sufficient basis for changing the attitudes towards PBR. Conducting the same analyses on data from existing experiments as well as new replications with the purpose of evaluating differences among perspectives will bring more clarity into the advantages and disadvantages of PBR techniques, and also give a better control over the validity threats.

14.9 Data on Individual Performance

Table 14.6 Data for each subject

Id	Perspective	Document	Time	Defects	Efficiency	Rate
1	U	ATM	187	8	2.567	0.276
2	D	PG	150	8	3.200	0.267
3	T	ATM	165	9	3.273	0.310
4	U	PG	185	11	3.568	0.367
5	D	ATM	155	8	3.097	0.276
6	T	PG	121	8	3.967	0.267
7	U	ATM	190	7	2.211	0.241
8	D	PG	260	7	1.615	0.233
9	T	ATM	123	6	2.927	0.207
10	U	PG	155	6	2.323	0.200
11	D	ATM	210	11	3.143	0.379
12	T	PG	88	9	6.136	0.300
13	U	ATM	280	11	2.357	0.379
14	D	PG	145	11	4.552	0.367
15	T	ATM	170	5	1.765	0.172
16	U	PG	120	6	3.000	0.200
17	D	ATM	190	9	2.842	0.310
18	T	PG	97	5	3.093	0.167
19	U	ATM	295	2	0.407	0.069
20	D	PG	180	7	2.333	0.233
21	T	ATM	306	7	1.373	0.241
22	U	PG	223	4	1.076	0.133
23	D	ATM	157	6	2.293	0.207
24	T	PG	130	6	2.769	0.200
25	U	ATM	195	13	4.000	0.448
26	D	PG	200	7	2.100	0.233
27	T	ATM	195	8	2.462	0.276
28	U	PG	125	5	2.400	0.167
29	D	ATM	200	8	2.400	0.276
30	T	PG	150	5	2.000	0.167

14.10 Data on Defects Found by Perspectives

14.10.1 PG Document

Table 14.7 Defects id D# found (1) or not found (0) by individuals reading the PG document

	Individuals																	
	User perspective						**Tester perspective**						**Designer perspective**					
D#	2	8	14	20	26	S	4	10	16	22	28	S	6	12	18	24	30	S
1	0	0	1	0	1	2	1	0	0	0	0	1	1	0	0	1	0	2
2	1	1	1	0	1	4	1	0	1	0	0	2	0	1	1	1	0	3
3	0	0	0	0	0	0	1	1	0	1	0	3	0	0	0	1	0	1
4	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1
5	0	0	1	1	1	3	1	0	0	0	1	2	0	1	1	0	0	2
6	1	1	0	0	0	2	0	1	1	0	0	2	1	0	0	1	1	3
7	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	1
8	1	1	1	1	1	5	1	1	1	1	0	4	1	1	0	1	0	3
9	1	1	1	1	0	4	1	1	1	0	0	3	0	1	0	0	0	1
10	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
11	1	1	0	1	1	4	0	0	0	0	0	0	0	1	1	0	0	2
12	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	1	1	1	1	0	1	1	4	1	1	0	1	1	4
15	0	0	1	0	0	1	0	1	0	0	1	2	1	0	0	0	0	1
16	1	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	2
17	0	0	0	1	1	2	1	0	0	0	0	1	0	1	0	0	0	1
18	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	1	0	0	1	0	0	0	0	1	1	1	0	1	0	1	3
22	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	1
23	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
24	0	0	1	1	0	2	0	0	0	0	1	1	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
28	0	1	1	0	0	2	0	0	0	0	0	0	1	0	0	0	0	1
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	1	0	1	0	0	2	0	0	1	0	0	1	0	0	0	0	0	0
S	8	7	11	7	7	40	11	6	6	4	5	32	8	9	5	6	5	33

14.10.2 ATM Document

Table 14.8 Defects if D# found (1) or not found (0) by individuals reading the ATM document

	Individuals																	
	User perspective						**Tester perspective**						**Designer perspective**					
D#	1	7	13	19	25	S	3	9	15	21	27	S	5	11	17	23	29	S
1	0	0	0	1	0	1	0	0	0	1	0	1	1	1	0	0	1	3
2	1	0	1	0	0	2	1	0	1	1	1	4	1	1	1	0	1	4
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	2
4	0	1	1	1	0	3	1	1	0	0	0	2	1	1	1	0	1	4
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	2
7	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	1	2
8	0	0	1	0	0	1	0	1	0	1	1	3	1	1	0	0	1	3
9	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	2
10	0	1	1	0	0	2	0	0	0	0	0	0	0	1	0	0	0	1
11	1	0	0	0	0	1	0	1	0	1	1	3	1	0	1	0	0	2
12	1	1	1	0	0	3	0	0	1	1	1	3	0	1	0	0	0	1
13	1	0	1	0	0	2	0	0	0	0	0	0	0	1	1	1	0	3
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	1	0	0	0	1	1	0	0	0	1	2	0	0	0	0	0	0
16	0	1	1	0	0	2	1	1	1	0	0	3	0	1	1	0	0	2
17	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1
18	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1
19	1	0	0	0	0	1	1	1	0	0	0	2	0	0	0	1	0	1
20	0	0	1	0	0	1	0	0	1	1	0	2	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
23	0	0	0	0	0	0	0	0	1	0	1	2	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	1	0	1	0	2	0	0	0	0	0	0
27	1	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	3
28	1	0	1	0	0	2	1	0	0	0	0	1	1	0	1	0	1	3
29	1	1	1	0	0	3	1	0	0	0	1	2	0	0	0	1	0	1
S	8	7	11	2	0	28	8	6	5	7	8	34	8	11	9	6	8	42

Acknowledgments First of all, the authors would like to thank the students who participated as subjects in the experiment. We would also like to give a special acknowledgement to Forrest Shull at University of Maryland who provided support on the UMD lab-pack and gave many good comments on a draft version of this paper. We are also grateful for all constructive comments made by the anonymous reviewers. Thanks also to Claes Wohlin, Martin Höst and Håkan Petersson at Dept. of Communication Systems, Lund University, who have carefully reviewed this paper. Special thanks to Anders Holtsberg at Centre for Mathematical Sciences, Lund University, for his expert help on statistical analysis. This work is partly funded by the National Board of Industrial and Technical Development (NUTEK), Sweden, grant 1K1P-97-09690.

Appendix A
Training Exercises

Explanations of the different types of exercises can be found in the Preface. In summary, the objective is to provide four types of exercises:

Understanding	These exercises aim at highlighting the most important issues from each chapter. Exercises are available in Chaps. 1–12.
Training	The objective of these exercises is to encourage practicing experimentation. This includes setting up hypotheses and performing the statistical analysis.
Reviewing	Chapters 13 and 14 include examples of experiments. The intention of this part is to provide help in reviewing and reading published experiments.
Assignments	These exercises are formulated to promote an understanding of how experiments can be used in software engineering to evaluate methods and techniques.

The understanding type questions can be found at the end of each chapter, while the three other types of exercises can be found in this appendix.

A.1 Training

The exercises are preferably solved either using a statistical program package or tables from books in statistics. The tables in Appendix B may be used, but the tables provided are only for the 5% significance level, so if other significance levels are used then other sources must be used. It should be remembered that Appendix B has primarily been provided to explain the examples in Chap. 11. The data in an Excel file and the answers to the training exercises can be found through the following link: https://portal.research.lu.se/en/publications/experimentation-in-software-engineering-2024-edition.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3

A.1.1 Normally Distributed Data

Probably the most complicated example of the statistical methods in Chap. 11 is the goodness of fit test for the normal distribution (see Section 11.3.13). Thus, it is appropriate to ensure a good understanding of that test.

1. Carry out the goodness of fit test, on the same data (see Table 11.20), using 12 segments instead.

A.1.2 Experience

In Chap. 13, the outcome of the Personal Software Process course is compared with the background of the students taking the course. The analysis conducted in Chap. 13 is only partial. The full data set is provided in Tables A.2 and A.3. In Table A.1, the survey material handed out at the first lecture is presented. The outcome of the survey is presented in Table A.2. The outcome of the PSP course is presented in Table A.3, where the following seven measures have been used to measure the outcome of the course:

Size	The number of new and changed lines of code for the 10 programs.
Time	The total development time for the 10 programs.
Prod.	The productivity measured as number of lines of code per development hour.
Faults	The number of faults logged for the 10 programs. This includes all faults found, for example, including compilation faults.
Faults/KLOC	The number of faults for each 1000 lines of code.
Pred. Size	The absolute relative error in predicting program size. The figures show the error in absolute percentages; for example, both over- and underestimates with 20% are shown as 20 without any sign indicating the direction of the estimation error.
Pred. Time	The absolute relative error in predicting the development time.

Based on the presentation in Chap. 13 and the data in Tables A.2 and A.3 answer the following questions.

1. How can the survey be improved? Think about what constitutes good measures of background, experience, and ability.
2. Define hypotheses, additional to those in Chap. 13, based on the available data. Discuss why these hypotheses are interesting.
3. What type of sampling has been used?
4. Analyze the hypotheses you have stated. What are the results?
5. Discuss the external validity of your findings. Can the results be generalized outside the PSP? Can the results be generalized to industrial software engineers?

Table A.1 Student characterization

Area	Description	Answer
Study program (denoted Line)	Answer: Computer Science and Engineering or Electrical Engineering	
General knowledge in computer science and software engineering (denoted SE)	1. Little, but curious about the new course 2. Not my speciality (focus on other subjects) 3. Rather good, but not my main focus (one of a couple of areas) 4. Main focus of my studies	
General knowledge in programming (denoted Prog.)	1. Only 1–2 courses 2. Three or more courses, no industrial experience 3. A few courses and some industrial experience 4. More than 3 courses and more than 1 year industrial experience	
Knowledge about the PSP (denoted PSP)	1. What is it? 2. I have heard about it 3. A general understanding of what it is 4. I have read some material	
Knowledge in C (denoted C)	1. No prior knowledge 2. Read a book or followed a course 3. Some industrial experience (less than 6 months) 4. Industrial experience	
Knowledge in C++ (denoted C++)	1. No prior knowledge 2. Read a book or followed a course 3. Some industrial experience (less than 6 months) 4. Industrial experience	
Number of courses (denoted Courses)	A list of courses was provided and the students were asked to put down a yes or no for whether they had taken the course or not. Moreover, they were asked to complement the list of courses if they had read something else they thought was a particularly relevant course.	

Table A.2 Information from background survey

Subject	Line	SE	Prog.	PSP	C	C++	Courses
1	1	2	1	2	1	1	2
2	1	3	2	1	2	1	4
3	2	3	2	2	2	2	7
4	1	3	2	3	2	1	3
5	1	3	2	3	2	1	5
6	2	4	3	2	1	1	7
7	2	3	2	2	1	2	7
8	1	3	2	2	1	1	4
9	2	4	3	2	1	1	9
10	2	4	2	1	1	1	7
11	1	2	2	1	2	1	3
12	2	4	3	2	1	1	9
13	2	4	3	2	3	3	8
14	2	3	2	2	1	1	6
15	1	3	2	2	1	1	5
16	2	4	2	1	1	1	10
17	1	3	3	1	1	1	5
18	2	4	3	2	1	3	6
19	2	4	3	3	3	3	8
20	1	1	1	1	1	1	2
21	2	3	3	2	2	2	10
22	2	3	2	3	1	1	5
23	1	3	2	2	1	1	4
24	1	2	1	1	1	1	3
25	2	4	3	1	2	2	7
26	1	3	2	2	1	1	5
27	2	4	3	2	3	2	7
28	1	3	2	3	1	1	2
29	2	4	2	3	1	1	7
30	2	3	3	1	2	3	6
31	1	3	2	2	2	2	5
32	2	3	3	1	2	2	10
33	2	4	3	1	1	1	5
34	1	2	2	1	2	2	3
35	1	2	1	1	1	1	2
36	1	2	1	2	1	1	2
37	1	2	2	2	2	2	2
38	2	4	2	2	2	1	6
39	1	2	1	2	1	1	2
40	2	4	3	1	4	4	7

(continued)

Table A.2 (continued)

Subject	Line	SE	Prog.	PSP	C	C++	Courses
41	2	3	3	2	2	2	8
42	2	4	3	2	2	2	9
43	1	3	2	1	1	1	3
44	1	4	3	2	3	2	7
45	2	4	2	2	2	1	6
46	2	2	4	2	4	4	7
47	2	4	3	2	3	2	7
48	1	2	2	2	1	1	2
49	1	3	3	1	1	1	3
50	2	3	2	3	1	1	8
51	2	4	2	4	2	2	8
52	2	4	3	3	3	2	8
53	2	4	3	3	2	2	10
54	1	2	1	2	1	1	2
55	1	2	2	2	1	1	4
56	2	3	2	1	1	1	8
57	1	2	3	1	1	1	4
58	2	4	3	3	1	1	6
59	1	2	2	2	2	1	4

A.1.3 Programming

In an experiment, 20 programmers have developed the same program, where 10 of them have used programming language A and 10 have used language B. Language A is newer and the company is planning to change to language A if it is better than language B. During the development, the size of the program, the development time, the total number of removed defects, and the number of defects removed in the test have been measured.

The programmers have been randomly assigned a programming language and the objective of the experiment is to evaluate if the language has any effect on the four measured variables. The collected data can be found in Table A.4. The data is fictitious.

1. Which design has been used in the experiment?
2. Define the hypotheses for the evaluation.
3. Use box plots to investigate the differences between the languages in terms of central tendency and dispersion with respect to all four factors. Is there any outlier and if so should it be removed?
4. Assume that parametric tests can be used. Evaluate the effect of the programming language on the four measured variables. Which conclusions can be drawn from the results?

Table A.3 Outcome from the PSP course

Subject	Size	Time	Prod.	Faults	Faults/ KLOC	Pred. Size	Pred. Time
1	839	3657	13.8	53	63.2	39.7	20.2
2	1249	3799	19.7	56	44.8	44.1	21.2
3	968	1680	34.6	71	73.3	29.1	25.1
4	996	4357	13.7	35	35.1	24.3	18.0
5	794	2011	23.7	32	40.3	26.0	13.2
6	849	2505	20.3	26	30.6	61.1	48.2
7	1455	4017	21.7	118	81.1	36.5	34.7
8	1177	2673	26.4	61	51.8	34.6	32.5
9	747	1552	28.9	41	54.9	51.0	18.2
10	1107	2479	26.8	59	53.3	22.6	14.0
11	729	3449	12.7	27	37.0	26.9	52.0
12	999	3105	19.3	63	63.1	26.0	19.8
13	881	2224	23.8	44	49.9	47.9	39.9
14	730	2395	18.3	94	128.8	63.0	20.3
15	1145	3632	18.9	70	61.1	33.3	34.8
16	1803	3193	33.9	98	54.4	52.9	21.8
17	800	2702	17.8	60	75.0	34.3	26.7
18	1042	2089	29.9	64	61.4	49.3	41.5
19	918	3648	15.1	43	46.8	49.7	71.5
20	1115	6807	9.8	26	23.3	34.1	22.4
21	890	4096	13.0	108	121.3	19.3	34.8
22	1038	3609	17.3	98	94.4	21.4	52.0
23	1251	6925	10.8	498	398.1	21.8	34.1
24	623	4216	8.9	53	85.1	40.5	36.3
25	1319	1864	42.5	92	69.7	43.7	45.0
26	800	4088	11.7	74	92.5	42.6	36.2
27	1267	2553	29.8	88	69.5	53.0	30.1
28	945	1648	34.4	42	44.4	33.3	17.9
29	724	4144	10.5	49	67.7	32.8	17.8
30	1131	2869	23.7	102	90.2	29.2	15.5
31	1021	2235	27.4	49	48.0	18.0	25.0
32	840	3215	15.7	69	82.1	85.6	54.0
33	985	5643	10.5	133	135.0	27.3	31.0
34	590	2678	13.2	33	55.9	83.0	20.0
35	727	4321	10.1	48	66.0	17.0	22.7
36	955	3836	14.9	76	79.6	33.3	36.8
37	803	4470	10.8	56	69.7	18.2	27.7
38	684	1592	25.8	28	40.9	35.0	34.1
39	913	4188	13.1	45	49.3	25.3	27.5
40	1200	1827	39.4	61	50.8	31.6	20.9

(continued)

Table A.3 (continued)

Subject	Size	Time	Prod.	Faults	Faults/ KLOC	Pred. Size	Pred. Time
41	894	2777	19.3	64	71.6	21.3	22.4
42	1545	3281	28.3	136	88.0	35.0	16.1
43	995	2806	21.3	71	71.4	15.6	38.3
44	807	2464	19.7	65	80.5	43.3	26.4
45	1078	2462	26.3	55	51.0	49.1	51.6
46	944	3154	18.0	71	75.2	59.0	39.2
47	868	1564	33.3	50	57.6	50.4	45.2
48	701	3188	13.2	31	44.2	21.2	49.7
49	1107	4823	13.8	86	77.7	19.3	28.4
50	1535	2938	31.3	71	46.3	29.6	20.7
51	858	7163	7.2	97	113.1	58.4	32.9
52	832	2033	24.6	84	101.0	48.4	25.6
53	975	3160	18.5	115	117.9	29.5	31.5
54	715	3337	12.9	40	55.9	41.7	26.6
55	947	4583	12.4	99	104.5	41.0	22.3
56	926	2924	19.0	77	83.2	32.5	34.7
57	711	3053	14.0	78	109.7	22.8	14.3
58	1283	7063	10.9	186	145.0	46.5	26.6
59	1261	3092	24.5	54	42.8	27.4	45.3

5. Evaluate the effect of the programming language on the four measured variables using a non-parametric test. Which conclusions can be drawn from the results? Compare the results to those achieved when using parametric tests.
6. Discuss the validity of the results and if it is appropriate to use a parametric test.
7. Assume that the participating programmers have chosen the programming language themselves. What consequences does this have for the validity of the results? Do the conclusions still hold?

A.1.4 Design

This exercise is based on data obtained from an experiment carried out by Briand, Bunse, and Daly. The experiment is further described by Briand et al. [40].

An experiment is designed to evaluate the impact of quality object-oriented design principles when intending to modify a given design. The quality design principles evaluated are the principles provided by Coad and Yourdon [47]. In the experiment two systems are used with one design for each system. One of the designs is a "good" design made using the design principles and the other is a "bad" design not using the principles. The two designs are documented in the same way in terms of layout and content and are of the same size, i.e., they are developed to

Table A.4 Data for programming exercise

Programming language	Program size (LOC)	Development time (minutes)	Total number of defects	Number of test defects
A	1408	3949	89	23
A	1529	2061	69	16
A	946	3869	170	41
A	1141	5562	271	55
A	696	5028	103	39
A	775	2296	75	29
A	1205	2980	79	11
A	1159	2991	194	28
A	862	2701	67	27
A	1206	2592	77	15
B	1316	3986	68	20
B	1787	4477	54	10
B	1105	3789	130	23
B	1583	4371	48	13
B	1381	3325	133	29
B	944	5234	80	25
B	1492	4901	64	21
B	1217	3897	89	29
B	936	3825	57	20
B	1441	4015	79	18

be as similar as possible except for following or not following the design principles. The objective of the experiment is to evaluate if the quality design principles ease impact analysis when identifying changes in the design.

The task for each participant is to undertake two separate impact analyses, one for each system design. The impact analysis entails marking all places in the design that have to be changed but not actually changing them . The first impact analysis is for a changed customer requirement and the second is for an enhancement in the systems functionality. Four measures are collected during the task:

Mod_Time: Time spent on identifying places for modification

Mod_Comp: Represents the completeness of the impact analysis and is defined as:

$$\text{Mod\_Comp} = \frac{\text{Number of correct places found}}{\text{Total number of places to be found}}$$

Mod_Corr: Represents the correctness of the impact analysis and is defined as:

$$\text{Mod\_Corr} = \frac{\text{Number of correct places found}}{\text{Total number of places indicated as found}}$$

Mod_Rate: The number of correct places found per time unit, that is:

$$\text{Mod\_Rate} = \frac{\text{Number of correct places found}}{\text{Time for identification}}$$

The experiment is conducted on two occasions to let each participant work with both the good design and the bad design. The subjects were randomly assigned to one of two groups, A or B. Group A worked with the good design on the first occasion and the bad design on the second. Group B studied the bad design first and then the good design. The collected data can be found in Table A.5.

1. Which design has been used in the experiment?
2. Define the hypotheses for the evaluation.
3. How should the missing values in Table A.5 be treated?
4. Assume that parametric tests can be used. Evaluate the effect of the quality design principles on the four measured variables. Which conclusions can be drawn from the results?
5. Evaluate the effect of the quality design principles on the four measured variables using non-parametric tests. Which conclusions can be drawn from the results? Compare the results to those achieved when using parametric tests.
6. Discuss the validity of the results and if it is appropriate to use parametric tests.
7. The participants in the experiment are students taking a software engineering course that have volunteered to be subjects. From which population is the sample taken? Discuss how this type of sampling will affect the external validity of the experiment. How can the sampling be made differently?

A.1.5 Inspections

This exercise refers to the example experiment in Chap. 14.

1. Rewrite the abstract in Chap. 14 to be a structured abstract, as defined in Chap. 12.
2. Conduct the scoping and planning steps for an *exact* replication of the experiment. In particular, define how many subjects should be enrolled to achieve a given level of confidence in the analysis.
3. Conduct the scoping step for a *differentiated* replication of the experiment. Define three different goal templates for three alternative replications. Discuss pros and cons of each alternative with respect to costs, risks, and gains (see also Figure 2.2).

Table A.5 Data for design exercise

		Good Object-Oriented design				Bad Object-Oriented design			
Participant	**Group**	**Mod_Time**	**Mod_Comp**	**Mod_Corr**	**Mod_Rate**	**Mod_Time**	**Mod_Comp**	**Mod_Corr**	**Mod_Rate**
P01	B	–	0.545	0.75	–	–	0.238	0.714	–
P02	B	–	0.818	1	–	–	0.095	1	–
P03	A	20	0.409	1	0.45	25	0.19	1	0.16
P04	B	22	0.818	1	0.818	25	0.238	1	0.2
P05	B	30	0.909	1	0.667	35	0.476	0.909	0.286
P07	A	–	0	–	–	38	0.476	1	0.263
P09	A	–	0.455	1	–	–	0.476	1	–
P10	B	–	0.409	0.9	–	–	0.381	1	–
P11	A	45	0.545	0.923	0.267	50	0.714	1	0.3
P12	B	–	0.773	1	–	–	0.714	1	–
P13	A	40	0.773	1	0.425	40	0.762	1	0.4
P14	B	30	0.909	1	0.667	30	0.333	0.875	0.233
P15	B	–	0.864	1	–	40	0.238	1	0.125
P16	B	30	0.773	1	0.567	–	–	–	–
P17	B	–	0.955	1	–	–	0.286	0.75	–
P18	B	–	0	–	–	–	0.19	1	–
P19	A	29	0.818	1	0.621	27	0.667	1	0.519
P20	A	9	0.591	1	1.444	15	0.19	0.8	0.267
P21	B	20	0.591	1	0.65	35	0.19	1	0.114
P22	B	30	0.682	1	0.5	20	0.714	1	0.75
P23	B	–	0.818	1	–	–	0.476	1	–
P24	A	30	0.773	1	0.567	40	0.762	1	0.4
P25	A	–	0.955	1	–	–	0.667	0.875	–
P26	B	25	0	0	0	25	0.095	0.5	0.08
P27	A	27	0.773	0.944	0.63	36	0.389	0.7	0.194
P28	A	25	0.773	1	0.68	30	0.667	1	0.467
P29	B	44	0.773	1	0.386	23	0.762	1	0.696
P31	A	–	0.409	1	–	–	0.286	0.75	–
P32	A	30	0.909	1	0.667	–	0.5	1	–
P33	A	65	0.818	1	0.277	–	0.619	1	–
P34	A	50	0.636	0.933	0.28	30	0.4	0.889	0.267
P35	A	10	0.591	1	1.3	10	0.667	1	1.4
P36	A	13	1	1	1.692	–	0.619	1	–

A.2 Reviewing

Below is a list of questions which are important to consider when reading or reviewing an article presenting an experiment. Use the list and review the examples presented in Chaps. 13 and 14, and also an experiment presented in the literature.

The list below should be seen as a checklist in addition to normal questions when reading an article. An example of a normal question could be: Is the abstract a good description of the content of the paper? Some specific aspects to consider when reading an experiment article are as follows:

- Is the experiment understandable and interesting in general?
- Does the experiment have any practical value?
- Are other experiments addressing the problem summarized and referenced?
- What is the population in the experiment?
- Is the sample used representative of the population?
- Are the dependent and independent variables clearly defined?
- Are the hypotheses clearly formulated?
- Is the type of design clearly stated?
- Is the design correct?
- Is the instrumentation described properly?
- Is the validity of the experiment treated carefully and convincingly?
- Are different types of validity threats addressed properly?
- Has the data been validated?
- Is the statistical power sufficient and are there enough subjects in the experiment?
- Are the appropriate statistical tests applied? Are parametric or non-parametric tests used and are they used correctly?
- Is the significance level used appropriate?
- Is the data interpreted correctly?
- Are the conclusions correct?
- Are the results not overstated?
- Is it possible to replicate the study?
- Is data provided?
- Is it possible to use the results for performing a meta-analysis?
- Is further work and experimentation in the area outlined?

A.3 Assignments

These assignments are based on the following general scenario. A company would like to improve their way of working by changing the software process. You are consulted as an expert in evaluating new techniques and methods in relation to the existing process. The company would like to know whether or not to change their software process.

You are expected to search for appropriate literature, review the existing literature on the subject, apply the experiment process, and write a report containing a recommendation for the company. The recommendation should discuss both the results of the experiment and other relevant issues for taking the decision whether or not to change the process. Other relevant issues include costs and benefits for making the change. If you are unable to find the correct costs, you are expected to make estimates. The latter may be in terms of relative costs.

The assignments are intentionally fairly open-ended to allow for interpretation and discussion. Each assignment is described in terms of prerequisites needed to perform the assignment and then the actual task is briefly described. It should be noted that the assignments below are examples of possible experiments that can be conducted. The important issue to bear in mind is that the main objective is that the assignments should provide practice in using experiments as part of an evaluation procedure.

Finally, it should be noted that some organizations provide what is called lab packages that can be used to replicate experiments. Lab packages are important as they allow us to build upon work by others and hence hopefully come to more generally valid results by replication. Some lab packages can be found by a search on the Internet. It may also be beneficial to contact the original experimenter to get support and perhaps also a non-published lab package.

A.3.1 Unit Test and Code Reviews

The company wants to evaluate if it is cost-effective to introduce code reviews. Unit testing is done today, although on non-reviewed code. Is this the best way to do it?

Prerequisites

- Suitable programs with defects that can be found during either reviews or testing.
- A review method, which may be ad hoc, but preferably it should be something more realistic, for example, a checklist-based approach. In this case, a checklist is needed.
- A testing method, which also may be ad hoc, but preferably it is based on, for example, usage or equivalence partitioning.

Task

- Evaluate if it is cost-effective to introduce code reviews.

A.3.2 Inspection Methods

Several different ways of conducting reviews are available. The company intends to introduce the best inspection method out of two possible choices. Which of the two methods is the best to introduce for the company?

Prerequisites

- Suitable software artifacts to review should be available.
- Two review methods with appropriate support in terms of, for example, checklists or description of different reading perspectives (see also Appendix A.1.5).

Task

- Assume that the company intends to introduce reviews of the chosen software artifacts. Which method should they introduce? Determine which of the inspection methods is best at finding defects. Is the best method also cost-effective?

A.3.3 Requirements Notation

It is important to write requirements specification so that all readers interpret them easily and in the same way. The company has several different notations to choose from. Which is the best way of representing requirements?

Prerequisites

- A requirements specification written in several different notations, for example, natural language and different graphical representations.

Task

- Evaluate if it is beneficial to change the company's notation for requirements specifications. Assume that the company uses natural language today.

Appendix B
Statistical Tables

This appendix contains statistical tables for a significance level of 5%. More elaborated tables can be found in most books on statistics, for example [166], and tables are also available on the Internet. The main objective here is to provide some information, so that the tests that are explained in Chap. 11 become understandable and so that the examples provided can be followed. This is important even if statistical packages are used for the calculations, since it is important to understand the underlying calculations before just applying the different statistical tests. It is also worth noting that the tables are a shortcut; for example, the values for the t-test, F-test, and Chi-square can be calculated from the respective distributions .

The following statistical tables are included:

- *t*-test (see Sects. 11.3.5 and 11.3.8 and Table B.1)
- Chi-square (see Sect. 11.3.13 and Table B.2)
- Mann-Whitney (see Sect. 11.3.6 and Table B.3)
- Wilcoxon (see Sect. 11.3.9 and Table B.4)
- F-test (see Sects. 11.3.7 and 11.3.11 and Table B.5)

Please note that in Table B.3, N_A is for the smaller sample and N_B for the larger sample.

Please note that Table B.5 provides the upper 0.025 percentage point of the F distribution with f_1 and f_2 being the degrees of freedom. This is equivalent to $F_{0.0025, f_1, f_2}$.

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3

Table B.1 Critical values two-tailed *t*-test (5%); see Sects. 11.3.5 and 11.3.8

Degrees of freedom	*t*-value
1	12.706
2	4.303
3	3.182
4	2.776
5	2.571
6	2.447
7	2.365
8	2.306
9	2.262
10	2.228
11	2.201
12	2.179
13	2.160
14	2.145
15	2.131
16	2.120
17	2.110
18	2.101
19	2.093
20	2.086
21	2.080
22	2.074
23	2.069
24	2.064
25	2.060
26	2.056
27	2.052
28	2.048
29	2.045
30	2.042
40	2.021
60	2.000
120	1.980
∞	1.960

Table B.2 Critical values one-tailed Chi-square test (5%); see Sect. 11.3.13

Degrees of freedom	χ^2
1	3.84
2	5.99
3	7.81
4	9.49
5	11.07
6	12.59
7	14.07
8	15.51
9	16.92
10	18.31
11	19.68
12	21.03
13	22.36
14	23.68
15	25.00
16	26.30
17	27.59
18	28.87
19	30.14
20	31.41
21	32.67
22	33.92
23	35.17
24	36.42
25	37.65
26	38.88
27	40.11
28	41.34
29	42.56
30	43.77
40	55.76
60	79.08
80	101.88
100	124.34

Table B.3 Critical values two-tailed Mann-Whitney (5%); see Sect. 11.3.6

N_B / N_A	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
3	0	1	1	2	2	3	3	4
4	1	2	3	4	4	5	6	7
5	2	3	5	6	7	8	9	11
6		5	6	8	10	11	13	14
7			8	10	12	14	16	18
8				13	15	17	19	22
9					17	20	23	26
10						23	26	29
11							30	33
12								37

Table B.4 Critical values two-tailed matched-pair Wilcoxon test (5%); see Sect. 11.3.9

n	**T**
6	0
7	2
8	3
9	5
10	8
11	10
12	13
13	17
14	21
15	25
16	29
17	34
18	40
19	46
20	52
22	66
25	89

Table B.5 Critical values two-tailed F-test (5%); see Sect. 11.3.7. For ANOVA, this is equivalent to a significance level of 2.5%; see Sect. 11.3.11

f_1 / f_2	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**15**	**20**	**30**	**40**	**60**	**120**	∞
1	648	800	864	900	922	937	948	957	963	969	977	985	993	1001	1006	1010	1014	1018
2	38.5	39.0	39.2	39.2	39.3	39.3	39.4	39.4	39.4	39.4	39.4	39.4	39.4	39.5	39.5	39.5	39.5	39.5
3	17.4	16.0	15.4	15.1	14.9	14.7	14.6	14.5	14.5	14.4	14.3	14.2	14.2	14.1	14.0	14.0	14.0	13.9
4	12.2	10.6	9.98	9.60	9.36	9.20	9.07	8.98	8.90	8.84	8.75	8.66	8.56	8.46	8.41	8.36	8.31	8.26
5	10.0	8.43	7.76	7.39	7.15	6.98	6.85	6.76	6.68	6.62	6.52	6.43	6.33	6.23	6.18	6.12	6.07	6.02
6	8.81	7.26	6.60	6.23	5.99	5.82	5.70	5.60	5.52	5.46	5.37	5.27	5.17	5.07	5.01	4.96	4.90	4.85
7	8.07	6.54	5.89	5.52	5.29	5.12	4.99	4.90	4.82	4.76	4.67	4.57	4.47	4.36	4.31	4.25	4.20	4.14
8	7.57	6.06	5.42	5.05	4.82	4.65	4.53	4.43	4.36	4.30	4.20	4.10	4.00	3.89	3.84	3.78	3.73	3.67
9	7.21	5.71	5.08	4.72	4.48	4.32	4.20	4.10	4.03	3.96	3.87	3.77	3.67	3.56	3.51	3.45	3.39	3.33
10	6.94	5.46	4.83	4.47	4.24	4.07	3.95	3.85	3.78	3.72	3.62	3.52	3.42	3.31	3.26	3.20	3.14	3.08
12	6.55	5.10	4.47	4.12	3.89	3.73	3.61	3.51	3.44	3.37	3.28	3.18	3.07	2.96	2.91	2.85	2.79	2.72
15	6.20	4.76	4.15	3.80	3.58	3.41	3.29	3.20	3.12	3.06	2.96	2.86	2.76	2.64	2.59	2.52	2.46	2.40
20	5.87	4.46	3.86	3.51	3.29	3.13	3.01	2.91	2.84	2.77	2.68	2.57	2.46	2.35	2.29	2.22	2.16	2.09
30	5.57	4.18	3.59	3.25	3.03	2.87	2.75	2.65	2.57	2.51	2.41	2.31	2.20	2.07	2.01	1.94	1.87	1.79
40	5.42	4.05	3.46	3.13	2.90	2.74	2.62	2.53	2.45	2.39	2.29	2.18	2.07	1.94	1.88	1.80	1.72	1.64
60	5.29	3.93	3.34	3.01	2.79	2.63	2.51	2.41	2.33	2.27	2.17	2.06	1.94	1.82	1.74	1.67	1.58	1.48
120	5.15	3.80	3.23	2.89	2.67	2.52	2.39	2.30	2.22	2.16	2.05	1.94	1.82	1.69	1.61	1.53	1.43	1.31
∞	5.02	3.69	3.12	2.79	2.57	2.41	2.29	2.19	2.11	2.05	1.94	1.83	1.71	1.57	1.48	1.39	1.27	1.00

References

1. Ali, N.B., Petersen, K., Wohlin, C.: A systematic literature review on the industrial use of software process simulation. J. Syst. Software **97**, 65–85 (2014). https://doi.org/10.1016/j.jss.2014.06.059
2. American Psychological Association: Ethical principles of psychologists and code of conduct. Am. Psychologist 1524–1611 (1992)
3. Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., Chatzigeorgiou, A.: Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Inf. Software Technol. **106**, 201–230 (2019). https://doi.org/10.1016/j.infsof.2018.10.006
4. Amschler Andrews, A., Pradhan, A.S.: Ethical issues in empirical software engineering: the limits of policy. Empirical Software Eng. **6**(2), 105–110 (2001). https://doi.org/10.1023/A:1011442319273
5. Anastas, J.W.: Research Design for the Social Work and the Human Services, 2nd edn. Columbia University Press, New York (2000). https://doi.org/10.7312/anas11890
6. Andersson, C., Runeson, P.: A spiral process model for case studies on software quality monitoring – method and metrics. Software Process Improv. Pract. **12**(2), 125–140 (2007). https://doi.org/10.1002/spip.311
7. Anthony, R.N.: Planning and Control Systems: A Framework for Analysis. Harvard University Graduate School of Business Administration, Boston (1965)
8. Arisholm, E., Gallis, H., Dybå, T., Sjøberg, D.I.K.: Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Trans. Software Eng. **33**(2), 65–86 (2007). https://doi.org/10.1109/TSE.2007.17
9. Association for Computing Machinery, ACM: Artifact review and badging – current (2020). https://www.acm.org/publications/policies/artifact-review-and-badging-current. Last accessed 20 Feb 2024
10. Auer, F., Ros, R., Kaltenbrunner, L., Runeson, P., Felderer, M.: Controlled experimentation in continuous experimentation: Knowledge and challenges. Inf. Software Technol. **134**, 106551 (2021). https://doi.org/10.1016/j.infsof.2021.106551
11. Avison, D., Baskerville, R., Myers, M.: Controlling action research projects. Inf. Technol. People **14**(1), 28–45 (2001). https://doi.org/10.1108/09593840110384762
12. Ayala, C., Turhan, B., Franch, X., Juristo, N.: Use and misuse of the term 'experiment' in mining software repositories research. IEEE Trans. Software Eng. **48**(11), 4229–4248 (2022). https://doi.org/10.1109/TSE.2021.3113558
13. Babbie, E.R.: Survey Research Methods. Wadsworth, Belmont (1998)

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3

14. Badampudi, D., Wohlin, C., Petersen, K.: Experiences from using snowballing and database searches in systematic literature studies. In: Proceedings International Conference on Evaluation and Assessment in Software Engineering (2015). https://doi.org/10.1145/2745802.2745818
15. Baltes, S., Ralph, P.: Sampling in software engineering research: a critical review and guidelines. Empirical Software Eng. **27**(4) (2022). https://doi.org/10.1007/s10664-021-10072-8
16. Basili, V.R.: Quantitative evaluation of software engineering methodology. In: Proceedings of the Pan Pacific Computer Conference, Melbourne, vol. 1, pp. 379–398 (1985)
17. Basili, V.R.: Software development: a paradigm for the future. In: Proceedings of the Annual International Computer Software & Applications Conference, pp. 471–485 (1989). https://doi.org/10.1109/CMPSAC.1989.65127
18. Basili, V.R.: The experimental paradigm in software engineering. In: Rombach, H.D., Basili, V.R., Selby, R.W. (eds.) Experimental Software Engineering Issues: Critical Assessment and Future Directives, no. 706 in Lecture Notes in Computer Science. Springer, Berlin (1993)
19. Basili, V.R.: Evolving and packaging reading technologies. J. Syst. Software **38**(1), 3–12 (1997). https://doi.org/10.1016/S0164-1212(97)00065-4
20. Basili, V., Green, S.: Software process evaluation at the SEL. IEEE Software, 58–66 (1994). https://doi.org/10.1109/52.300090
21. Basili, V.R., Rombach, H.: The TAME project: Towards improvement-oriented software environments. IEEE Trans. Software Eng. **14**(6) (1988). https://doi.org/10.1109/32.6156
22. Basili, V.R., Selby, R.W.: Comparing the effectiveness of software testing strategies. IEEE Trans. Software Eng. **13**(12), 1278–1298 (1987). https://doi.org/10.1109/TSE.1987.232881
23. Basili, V.R., Weiss, D.M.: A methodology for collecting valid software engineering data. IEEE Trans. Software Eng. **10**(6), 728–737 (1984). https://doi.org/10.1109/TSE.1984.5010301
24. Basili, V.R., Selby, R.W., Hutchens, D.H.: Experimentation in software engineering. IEEE Trans. Software Eng. **12**(7), 733–743 (1986). https://doi.org/10.1109/TSE.1986.6312975
25. Basili, V.R., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.V.: The empirical investigation of perspective-based reading. Empirical Software Eng. **1**(2), 133–164 (1996). https://doi.org/10.1007/BF00368702
26. Basili, V., Green, S., Laitenberger, O., Lanubile, F., Shull, F., Sørumgård, S., Zelkowitz, M.: Lab package for the empirical investigation of perspective-based reading. Technical report. Univeristy of Maryland (1998)
27. Basili, V.R., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Trans. Software Eng. **25**(4), 456–473 (1999). https://doi.org/10.1109/32.799939
28. Basili, V., Caldiera, G., Rombach, H., van Solingen, R.: Goal Question Metrics paradigm. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, pp. 528–532. Wiley, Hoboken (2002). https://doi.org/10.1002/0471028959.sof142
29. Basili, V.R., Caldiera, G., Rombach, H.: Experience factory. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, pp. 469–476. Wiley, Hoboken (2002). https://doi.org/10.1002/0471028959.sof110
30. Baskerville, R.: What design science is not. Eur. J. Inf. Syst. **17**(5), 441–443 (2008). https://doi.org/10.1057/ejis.2008.45
31. Baskerville, R.L., Wood-Harper, A.T.: A critical perspective on action research as a method for information systems research. Int. J. Inf. Technol. **11**(3), 235–246 (1996). https://doi.org/10.1080/026839696345289
32. Benbasat, I., Goldstein, D.K., Mead, M.: The case research strategy in studies of information systems. MIS Q. **11**(3), 369–386 (1987). https://doi.org/10.2307/248684
33. Bergman, B., Klefsjö, B.: Quality from Customer Needs to Customer Satisfaction. Studentlitteratur, Lund (2022)
34. Berndt, A.E.: Sampling methods. J. Hum. Lactation **36**(2), 224–226 (2020). https://doi.org/10.1177/0890334420906850
35. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qual. Res. Psychol. **3**(2), 77–101 (2006). https://doi.org/10.1191/1478088706qp063oa

36. Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Software **80**(4), 571–583 (2007). https://doi.org/10.1016/j.jss.2006.07.009
37. Brereton, P., Kitchenham, B., Budgen, D., Li, Z.: Using a protocol template for case study planning. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering. University of Bari, Bari (2008). https://doi.org/10.14236/ewic/EASE2008.5
38. Briand, L., El Emam, K., Morasca, S.: On the application of measurement theory in software engineering. Empirical Software Eng. **1**(1), 61–88 (1996). https://doi.org/10.1007/BF00125812
39. Briand, L.C., Differding, C.M., Rombach, H.: Practical guidelines for measurement-based process improvement. Software Process Improv. Pract. **2**(4), 253–280 (1996). https://doi.org/10.1002/(SICI)1099-1670(199612)2:4<253::AID-SPIP53>3.0.CO;2-G
40. Briand, L.C., Bunse, C., Daly, J.W.: A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. IEEE Trans. Software Eng. **27**(6), 513–530 (2001). https://doi.org/10.1109/32.926174
41. British Psychological Society: Ethical principles for conducting research with human participants. Psychologist, 33–35 (1993)
42. Budgen, D., Kitchenham, B.A., Charters, S.M., Turner, M., Brereton, P., Linkman, S.: Presenting software engineering results using structured abstracts: a randomised experiment. Empirical Software Eng. **13**, 435–468 (2008). https://doi.org/10.1007/s10664-008-9075-7
43. Budgen, D., Burn, A.J., Kitchenham, B.A.: Reporting computing projects through structured abstracts: a quasi-experiment. Empirical Software Eng. **16**(2), 244–277 (2011). https://doi.org/10.1007/s10664-010-9139-3
44. Campbell, D.T., Stanley, J.C.: Experimental and Quasi-Experimental Designs for Research. Ravenio Books (2015)
45. Ciolkowski, M., Differding, C., Laitenberger, O., Münch, J.: Empirical investigation of perspective-based reading: a replicated experiment. Technical Report 97-13, Fraunhofer Institute for Experimental Software Engineering (1997)
46. Clason, D.L., Dormody, T.J.: Analyzing data measured by individual likert-type items. J. Agric. Educ. **35**(4), 31–35 (1994). https://doi.org/10.5032/jae.1994.04031
47. Coad, P., Yourdon, E.: Object-Oriented Design, 1st edn. Yourdon Press, New Jersey (1991)
48. Cohen, J.: Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. Psychol. Bull. **70**, 213–220 (1968). https://doi.org/10.1037/h0026256
49. Collis, J., Hussey, R.: Business Research: A Practical Guide for Undergraduate and Postgraduate Students. Bloomsbury Publishing, London (2021)
50. Cook, T.D., Campbell, D.T.: Quasi-Experimentation – Design and Analysis Issues for Field Settings. Houghton Mifflin Company, Boston (1979)
51. Corbin, J., Strauss, A.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 3rd edn. SAGE, Newcastle upon Tyne (2008). https://doi.org/10.4135/9781452230153
52. Creswell, J.W., Creswell, J.D.: Research Design: Qualitative, Quantitative, and Mixed methods Approaches. Sage publications, Thousand Oaks (2017)
53. Cruzes, D.S., Dybå, T.: Research synthesis in software engineering: A tertiary study. Inf. Software Technol. **53**(5), 440–455 (2011). https://doi.org/10.1016/j.infsof.2011.01.004
54. Cruzes, D.S., Dybå, T., Runeson, P., Höst, M.: Case studies synthesis: A thematic, cross-case, and narrative synthesis worked example. Empirical Software Eng. **20**(6), 1634–1665 (2015). https://doi.org/10.1007/s10664-014-9326-8
55. Dalkey, N., Helmer, O.: An experimental application of the Delphi method to the use of experts. Manage. Sci. **9**(3), 458–467 (1963). https://doi.org/10.1287/mnsc.9.3.458
56. DeMarco, T.: Controlling Software Projects: Management, Measurement, and Estimates. Prentice Hall PTR, Hoboken (1986)
57. Demming, W.E.: Out of the Crisis. MIT Press, Cambridge (2018)

58. Dieste, O., Grimán, A., Juristo, N.: Developing search strategies for detecting relevant experiments. Empirical Software Eng. **14**, 513–539 (2009). https://doi.org/10.1007/s10664-008-9091-7
59. Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., Lindeberg, O.: Cooperative method development. Empirical Software Eng. **13**(3), 231–260 (2007). https://doi.org/10.1007/s10664-007-9057-1
60. Doolan, E.P.: Experiences with Fagan's inspection method. Software Pract. Exp. **22**(2), 173–182 (1992). https://doi.org/10.1002/spe.4380220205
61. Drisko, J.W., Maschi, T.: Content Analysis. Oxford University Press, Oxford (2016)
62. Dybå, T., Dingsøyr, T.: Strength of evidence in systematic reviews in software engineering. In: Proceedings International Symposium on Empirical Software Engineering and Measurement, pp. 178–187 (2008). https://doi.org/10.1145/1414004.1414034
63. Dybå, T., Kitchenham, B.A., Jørgensen, M.: Evidence-based software engineering for practitioners. IEEE Software **22**, 58–65 (2005). https://doi.org/10.1109/MS.2005.6
64. Dybå, T., Kampenes, V.B., Sjøberg, D.I.K.: A systematic review of statistical power in software engineering experiments. Inf. Software Technol. **48**(8), 745–755 (2006). https://doi.org/10.1016/j.infsof.2005.08.009
65. Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting empirical methods for software engineering research. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_11
66. Eick, S.G., Loader, C.R., Long, M.D., Votta, L.G., Vander Wiel, S.: Estimating software fault content before coding. In: Proceedings of the International Conference on Software Engineering, pp. 59–65 (1992). https://doi.org/10.1109/ICSE.1992.753490
67. Eisenhardt, K.M.: Building theories from case study research. Acad. Manage. Rev. **14**(4), 532 (1989). https://doi.org/10.2307/258557
68. Emerson, R.M., Fretz, R.I., Shaw, L.L.: Participant observation and fieldnotes. In: Stanley, L. (ed.) Handbook of Ethnography, pp. 352–368. SAGE Publications Ltd., Thousand Oaks (2001)
69. Endres, A., Rombach, H.D.: A Handbook of Software and Systems Engineering – Empirical Observations, Laws and Theories. Pearson Education, London (2003)
70. Engström, E., Storey, M.A., Runeson, P., Höst, M., Baldassarre, M.T.: How software engineering research aligns with design science: a review. Empirical Software Eng. **25**, 2630–2660 (2020). https://doi.org/10.1007/s10664-020-09818-7
71. Fagan, M.E.: Design and code inspections to reduce errors in program development. In: Broy, M., Denert, E. (eds.) Software Pioneers. Springer, Berlin (2002). https://doi.org/0.1007/978-3-642-59412-0_35
72. Fagerholm, F., Guinea, A.S., Mäenpää, H., Münch, J.: The RIGHT model for continuous experimentation. J. Syst. Software **123**, 292–305 (2017). https://doi.org/10.1016/j.jss.2016.03.034
73. Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A., Oivo, M.: Empirical software engineering experts on the use of students and professionals in experiments. Empirical Software Eng. **23**, 452–489 (2018). https://doi.org/10.1007/s10664-017-9523-3
74. Felderer, M., Travassos, G.H.: Contemporary Empirical Methods in Software Engineering. Springer, Cham (2020)
75. Fenton, N.: Software measurement: a necessary scientific basis. IEEE Trans. Software Eng. **3**(20), 199–206 (1994). https://doi.org/10.1109/32.268921
76. Fenton, N., Bieman, J.: Software Metrics: A Rigorous and Practical Approach, 2nd edn. CRC Press, Boca Raton (2014). https://doi.org/10.1201/b17461
77. Fenton, N., Pfleeger, S.L., Glass, R.L.: Science and substance: a challenge to software engineers. IEEE Software, 86–95 (1994). https://doi.org/10.1109/52.300094
78. Fink, A.: The Survey Handbook, 2nd edn. SAGE, Newcastle upon Tyne (2003)
79. Fitzgerald, B., Stol, K.J.: Continuous software engineering: A roadmap and agenda. J. Syst. Software **123**, 176–189 (2017). https://doi.org/10.1016/j.jss.2015.06.063

80. Flyvbjerg, B.: Five misunderstandings about case-study research. Qual. Inq. **12**(2), 219–245 (2006). https://doi.org/10.1177/1077800405284363
81. Frigge, M., Hoaglin, D.C., Iglewicz, B.: Some implementations of the boxplot. Am. Stat. **43**(1), 50–54 (1989). https://doi.org/10.1080/00031305.1989.10475612
82. Fusaro, P., Lanubile, F., Visaggio, G.: A replicated experiment to assess requirements inspection techniques. Empirical Software Eng. **2**(1), 39–57 (1997). https://doi.org/10.1023/A:1009742216007
83. García-Mireles, G.A., Morales-Trujillo, M.E.: Gamification in software engineering: a tertiary study. In: Proceedings of the International Conference on Software Process Improvement, pp. 116–128 (2020). https://doi.org/10.1007/978-3-030-33547-2_10
84. Garner, P., Hopewell, S., Chandler, J., MacLehose, H., Akl, E.A., Beyene, J., Chang, S., Churchill, R., Dearness, K., Guyatt, G., et al.: When and how to update systematic reviews: consensus and checklist. BMJ **354** (2016). https://doi.org/10.1136/bmj.i3507
85. Garousi, V., Felderer, M., Mäntylä, M.V.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Software Technol. **106**, 101–121 (2019). https://doi.org/10.1016/j.infsof.2018.09.006
86. Ghazi, A.N., Petersen, K., Reddy, S.S.V.R., Nekkanti, H.: Survey research in software engineering: problems and mitigation strategies. IEEE Access **7**, 24703–24718 (2018). https://doi.org/10.1109/ACCESS.2018.2881041
87. Glaser, B., Strauss, A.: Discovery of Grounded Theory: Strategies for Qualitative Research. Routledge, Milton Park (2017). https://doi.org/10.4324/9780203793206
88. Glass, R.L.: The software research crisis. IEEE Software, 42–47 (1994). https://doi.org/10.1109/52.329400
89. Glass, R.L., Vessey, I., Ramesh, V.: Research in software engineering: an analysis of the literature. Inf. Software Technol. **44**(8), 491–506 (2002). https://doi.org/10.1016/S0950-5849(02)00049-6
90. Gómez, O.S., Juristo, N., Vegas, S.: Replication types in experimental disciplines. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (2010). https://doi.org/10.1145/1852786.1852790
91. Gorschek, T., Wohlin, C.: Requirements abstraction model. Requir. Eng. **11**, 79–101 (2006). https://doi.org/10.1007/s00766-005-0020-7
92. Gorschek, T., Garre, P., Larsson, S., Wohlin, C.: A model for technology transfer in practice. IEEE Software **23**(6), 88–95 (2006). https://doi.org/10.1109/MS.2006.147
93. Gorschek, T., Garre, P., Larsson, S., Wohlin, C.: Industry evaluation of the requirements abstraction model. Requir. Eng. **12**, 163–190 (2007). https://doi.org/10.1007/s00766-007-0047-z
94. Grady, R.B., Caswell, D.L.: Software Metrics: Establishing a Company-Wide Program. Prentice-Hall, Hoboken (1994)
95. Grant, E.E., Sackman, H.: An exploratory investigation of programmer performance under on-line and off-line conditions. IEEE Trans. Hum. Factors Electron. **HFE-8**(1), 33–48 (1967). https://doi.org/10.1109/THFE.1967.233303
96. Gregor, S.: The nature of theory in information systems. MIS Q. **30**(3), 491–506 (2006). https://doi.org/10.2307/25148742
97. Guba, E.G.: Criteria for assessing the trustworthiness of naturalistic inquiries. ECTJ **29**(2), 75–91 (1981). https://doi.org/10.1007/BF02766777
98. Hall, T., Flynn, V.: Ethical issues in software engineering research: a survey of current practice. Empirical Software Eng. **6**, 305–317 (2001). https://doi.org/10.1023/A:1011922615502
99. Hall, T., Baddoo, N., Beecham, S., Robinson, H., Sharp, H.: A systematic review of theory use in studies investigating the motivations of software engineers. ACM Trans. Software Eng. Methodol. **18**(3), 1–29 (2009). https://doi.org/10.1145/1525880.1525883
100. Hannay, J.E., Sjøberg, D.I.K., Dybå, T.: A systematic review of theory use in software engineering experiments. IEEE Trans. Software Eng. **33**(2), 87–107 (2007). https://doi.org/10.1109/TSE.2007.12

101. Hannay, J.E., Dybå, T., Arisholm, E., Sjøberg, D.I.K.: The effectiveness of pair programming: a meta-analysis. Inf. Software Technol. **51**(7), 1110–1122 (2009). https://doi.org/10.1016/j.infsof.2009.02.001
102. Harwood, T.G., Garry, T.: An overview of content analysis. Mark. Rev. **3**(4), 479–498 (2003). https://doi.org/10.1362/146934703771910080
103. Hassan, A.E.: The road ahead for mining software repositories. In: Proceedings of the Frontiers of Software Maintenance, pp. 48–57 (2008). https://doi.org/10.1109/fosm.2008.4659248
104. Hayes, W.: Research synthesis in software engineering: a case for meta-analysis. In: Proceedings of the International Software Metrics Symposium, pp. 143–151 (1999). https://doi.org/10.1109/METRIC.1999.809735
105. Hetzel, B.: Making Software Measurement Work: Building an Effective Measurement Program. John Wiley and Sons, Hoboken (1993)
106. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004). https://doi.org/10.2307/25148625
107. Hoda, R.: Socio-technical grounded theory for software engineering. IEEE Trans. Software Eng. **48**(10), 3808–3832 (2022). https://doi.org/10.1109/TSE.2021.3106280
108. Hoda, R., Salleh, N., Grundy, J., Tee, H.M.: Systematic literature reviews in agile software development: a tertiary study. Inf. Software Technol. **85**, 60–70 (2017). https://doi.org/10.1016/j.infsof.2017.01.007
109. Höst, M., Regnell, B., Wohlin, C.: Using students as subjects – a comparative study of students and professionals in lead-time impact assessment. Empirical Software Eng. **5**(3), 201–214 (2000). https://doi.org/10.1023/A:1026586415054
110. Höst, M., Wohlin, C., Thelin, T.: Experimental context classification: incentives and experience of subjects. In: Proceedings of the International Conference on Software Engineering, pp. 470–478 (2005). https://doi.org/10.1145/1062455.1062539
111. Hove, S.E., Anda, B.: Experiences from conducting semi-structured interviews in empirical software engineering research. In: Proceedings of the 11th IEEE International Software Metrics Symposium, pp. 1–10 (2005). https://doi.org/10.1109/METRICS.2005.24
112. Humphrey, W.S.: Managing the Software Process. Addison-Wesley, Boston (1989)
113. Humphrey, W.S.: A Discipline for Software Engineering. Pearson Education, London (1995)
114. Humphrey, W.S.: Introduction to the Personal Software Process. Addison Wesley, Boston (1997)
115. IEEE: IEEE standard glossary of software engineering terminology. Technical Report. IEEE Std 610.12-1990, IEEE (1990)
116. Imtiaz, S., Bano, M., Ikram, N., Niazi, M.: A tertiary study: experiences of conducting systematic literature reviews in software engineering. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, pp. 177–182 (2013). https://doi.org/10.1145/2460999.2461025
117. Iversen, J.H., Mathiassen, L., Nielsen, P.A.: Managing risk in software process improvement: an action research approach. MIS Q. **28**(3), 395–433 (2004). https://doi.org/10.2307/25148645
118. Jalali, S., Wohlin, C.: Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 29–38 (2012). https://doi.org/10.1145/2372251.2372257
119. Jedlitschka, A., Pfahl, D.: Reporting guidelines for controlled experiments in software engineering. In: Proceedings of the International Symposium on Empirical Software Engineering, pp. 95–104 (2005). https://doi.org/10.1109/ISESE.2005.1541818
120. Jiménez, M., Piattini, M.: Problems and solutions in distributed software development: a systematic review. In: Proceedings of the Software Engineering Approaches for Offshore and Outsourced Development, pp. 107–125 (2009). https://doi.org/10.1007/978-3-642-01856-5_8
121. Johnson, P.M., Tjahjono, D.: Does every inspection really need a meeting? Empirical Software Eng. **3**(1), 9–35 (1998). https://doi.org/10.1023/A:1009787822215

122. Juristo, N., Moreno, A.M.: Basics of Software Engineering Experimentation. Springer-Verlag US, New York (2001)
123. Juristo, N., Vegas, S.: The role of non-exact replications in software engineering experiments. Empirical Software Eng. **16**, 295–324 (2011). https://doi.org/10.1007/s10664-010-9141-9
124. Kachigan, S.K.: Statistical Analysis: An Interdisciplinary Introduction to Univariate & Multivariate Methods. Radius Press, New York (1986)
125. Kachigan, S.K.: Multivariate Statistical Analysis – A Conceptual Introduction, 2nd edn. Radius Press, New York (1991)
126. Kalhoro, S., Rehman, M., Ponnusamy, V., Shaikh, F.B.: Extracting key factors of cyber hygiene behaviour among software engineers: a systematic literature review. IEEE Access **9**, 99339–99363 (2021). https://doi.org/10.1109/ACCESS.2021.3097144
127. Kamei, F., Wiese, I., Lima, C., Polato, I., Nepomuceno, V., Ferreira, W., Ribeiro, M., Pena, C., Cartaxo, B., Pinto, G., Soares, S.: Grey literature in software engineering: a critical review. Inf. Software Technol. **138**, 106609 (2021). https://doi.org/10.1016/j.infsof.2021.106609
128. Kampenes, V.B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K.: A systematic review of effect size in software engineering experiments. Inf. Software Technol. **49**(11–12), 1073–1086 (2007). https://doi.org/10.1016/j.infsof.2007.02.015
129. Karahasanović, A., Anda, B., Arisholm, E., Hove, S.E., Jørgensen, M., Sjøberg, D.I.K., Welland, R.: Collecting feedback during software engineering experiments. Empirical Software Eng. **10**(2), 113–147 (2005). https://doi.org/10.1007/s10664-004-6189-4
130. Karlström, D., Runeson, P., Wohlin, C.: Aggregating viewpoints for strategic software process improvement. IEE Proceed. Software **149**(5), 143–152 (2002). https://doi.org/10.1049/ip-sen:20020696
131. Kasunic, M.: Designing an effective survey. Technical report. Carnegie Mellon University, Software Engineering Institute Pittsburgh (2005). https://apps.dtic.mil/sti/pdfs/ADA441817.pdf
132. Kitchenham, B.: The role of replications in empirical software engineering – a word of warning. Empirical Software Eng. **13**, 219–221 (2008). https://doi.org/10.1007/s10664-008-9061-0
133. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (version 2.3). Techncial Report. EBSE Technical Report EBSE-2007-01, Keele University and Durham University (2007)
134. Kitchenham, B.A., Pfleeger, S.L.: Principles of survey research part 2: designing a survey. ACM SIGSOFT Software Eng. Notes **27**(1), 18–20 (2002). https://doi.org/10.1145/566493.566495
135. Kitchenham, B.A., Pfleeger, S.L.: Personal opinion surveys. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 63–92. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_3
136. Kitchenham, B., Pickard, L., Pfleeger, S.L.: Case studies for method and tool evaluation. IEEE Software, 52–62 (1995). https://doi.org/10.1109/52.391832
137. Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Software Eng. **28**(8), 721–734 (2002). https://doi.org/10.1109/TSE.2002.1027796
138. Kitchenham, B., Fry, J., Linkman, S.: The case against cross-over designs in software engineering. In: Proceedings of the International Workshop on Software Technology and Engineering Practice, pp. 65–67 (2003). https://doi.org/10.1109/STEP.2003.32
139. Kitchenham, B.A., Dybå, T., Jørgensen, M.: Evidence-based software engineering. In: Proceedings of the International Conference on Software Engineering, pp. 273–281 (2004). https://doi.org/10.1109/ICSE.2004.1317449
140. Kitchenham, B.A., Al-Khilidar, H., Babar, M.A., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., Zhu, L.: Evaluating guidelines for reporting empirical software engineering studies. Empirical Software Eng. **13**(1), 97–121 (2007). https://doi.org/10.1007/s10664-007-9053-5

141. Kitchenham, B.A., Jeffery, D.R., Connaughton, C.: Misleading metrics and unsound analyses. IEEE Software **24**, 73–78 (2007). https://doi.org/10.1109/MS.2007.49
142. Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering – a systematic literature review. Inf. Software Technol. **51**(1), 7–15 (2009). https://doi.org/10.1016/j.infsof.2008.09.009
143. Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O.P., Turner, M., Niazi, M., Linkman, S.: Systematic literature reviews in software engineering – a tertiary study. Inf. Software Technol. **52**(8), 792–805 (2010). https://doi.org/10.1016/j.infsof.2010.03.006
144. Kitchenham, B., Sjøberg, D.I.K., Brereton, O.P., Budgen, D., Dybå, T., Höst, M., Pfahl, D., Runeson, P.: Can we evaluate the quality of software engineering experiments? In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (2010). https://doi.org/10.1145/1852786.1852789
145. Kitchenham, B.A., Budgen, D., Brereton, O.P.: Using mapping studies as the basis for further research – a participant-observer case study. Inf. Software Technol. **53**(6), 638–651 (2011). https://doi.org/10.1016/j.infsof.2010.12.011
146. Kitchenham, B.A., Budgen, D., Brereton, P.: Evidence-Based Software Engineering and Systematic Reviews, vol. 4. CRC Press, Boca Raton (2015)
147. Kitchenham, B., Madeyski, L., Budgen, D.: SEGRESS: software engineering guidelines for reporting secondary studies. IEEE Trans. Software Eng. **49**(3), 1273–1298 (2023). https://doi.org/10.1109/TSE.2022.3174092
148. Klein, H.K., Myers, M.D.: A set of principles for conducting and evaluating interpretive field studies in information systems. MIS Q., 67–93 (1999). https://doi.org/10.2307/249410
149. Kontio, J., Bragge, J., Lehtola, L.: The focus group method as an empirical tool in software engineering. In: Guide to Advanced Empirical Software Engineering, pp. 93–116. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_4
150. Kotti, Z., Galanopoulou, R., Spinellis, D.: Machine learning for software engineering: a tertiary study. ACM Comput. Surv. **55**(12), 1–39 (2023). https://doi.org/10.1145/3572905
151. Krishnaiah, V., Narsimha, G., Chandra, N.S.: Survey of classification techniques in data mining. Int. J. Comput. Sci. Eng. **2**(9), 65–74 (2014)
152. Laitenberger, O., Atkinson, C., Schlich, M., El Emam, K.: An experimental comparison of reading techniques for defect detection in UML design documents. J. Syst. Software **53**(2), 183–204 (2000). https://doi.org/10.1016/S0164-1212(00)00052-2
153. Larsson, R.: Case survey methodology: quantitative analysis of patterns across case studies. Acad. Manage. J. **36**(6), 1515–1546 (1993). https://doi.org/10.5465/256820
154. Law, A.M., Kelton, W.D.: Simulation Modeling and Analysis, vol. 3. Mcgraw-Hill, New York (2007)
155. Lee, A.S.: A scientific methodology for MIS case studies. MIS Q. **13**(1), 33–50 (1989). https://doi.org/10.2307/248698
156. Lehman, M.M.: Program, life-cycles and the laws of software evolution. Proc. IEEE **68**(9), 1060–1076 (1980). https://doi.org/10.1109/PROC.1980.11805
157. Lethbridge, T.C., Sim, S.E., Singer, J.: Studying software engineers: data collection techniques for software field studies. Empirical Software Eng. **10**, 311–341 (2005). https://doi.org/10.1007/s10664-005-1290-x
158. Linåker, J., Sulaman, S.M., Höst, M., Maiani de Mello, R.: Guidelines for conducting surveys in software engineering. Technical report. Lund University (2015). https://lucris.lub.lu.se/ws/portalfiles/portal/6062997/5463412.pdf
159. Linger, R.: Cleanroom process model. IEEE Software **11**, 50–58 (1994). https://doi.org/10.1109/52.268956
160. Linkman, S., Rombach, H.D.: Experimentation as a vehicle for software technology transfer – a family of software reading techniques. Inf. Software Technol. **39**(11), 777–780 (1997). https://doi.org/10.1016/S0950-5849(97)00029-3
161. Lucas, W.A.: The case survey method: aggregating case experience. Technical Report. R-1515-RC, The RAND Corporation, Santa Monica (1974)

162. Lucas Jr, H.C., Kaplan, R.B.: A structured programming experiment. Comput. J. **19**(2), 136–138 (1976). https://doi.org/10.1093/comjnl/19.2.136
163. Lyu, M.R. (ed.): Handbook of Software Reliability Engineering. McGraw-Hill, New York (1996)
164. Maldonado, J.C., Carver, J., Shull, F., Fabbri, S., Dória, E., Martimiano, L., Mendonça, M., Basili, V.: Perspective-based reading: a replicated experiment focused on individual reviewer effectiveness. Empirical Software Eng. **11**, 119–142 (2006). https://doi.org/10.1007/s10664-006-5967-6
165. Manly, B.F.J.: Multivariate Statistical Methods – A Primer, 3rd edn. Chapman and Hall/CRC, Boca Raton (2004). https://doi.org/10.1201/b16974
166. Marascuilo, L.A., Serlin, R.C.: Statistical Methods for the Social and Behavioral Sciences. W H Freeman/Times Books/Henry Holt & Co, New York City (1988)
167. McGregor, S.L.T.: Ethical considerations in research about organizations: compendium of strategies. Ethics Prog. **14**(2), 4–23 (2023). https://doi.org/10.14746/eip.2023.2.1
168. Mendes, E., Wohlin, C., Felizardo, K., Kalinowski, M.: When to update systematic literature reviews in software engineering. J. Syst. Software **167**, 110607 (2020). https://doi.org/10.1016/j.jss.2020.110607
169. Mendez, D., Graziotin, D., Wagner, S., Seibold, H.: Open science in software engineering. In: Felderer, M., Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering, pp. 477–501. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-32489-6_17
170. Méndez Fernández, D., Passoth, J.H.: Empirical software engineering: from discipline to interdiscipline. J. Syst. Software **148**, 170–179 (2019). https://doi.org/10.1016/j.jss.2018.11.019
171. Méndez Fernández, D., Monperrus, M., Feldt, R., Zimmermann, T.: The open science initiative of the empirical software engineering journal. Empirical Software Eng. **24**(3), 1057–1060 (2019). https://doi.org/10.1007/s10664-019-09712-x
172. Miller, J.: Estimating the number of remaining defects after inspection. Softw. Test. Verif. Reliab. **9**(4), 167–189 (1999). https://doi.org/10.1002/(SICI)1099-1689(199909)9:3<167::AID-STVR185>3.0.CO;2-E
173. Miller, J.: Applying meta-analytical procedures to software engineering experiments. J. Syst. Software **54**(1), 29–39 (2000). https://doi.org/10.1016/S0164-1212(00)00024-8
174. Miller, J.: Statistical significance testing: a panacea for software technology experiments? J. Syst. Software **73**, 183–192 (2004). https://doi.org/10.1016/j.jss.2003.12.019
175. Miller, J.: Replicating software engineering experiments: a poisoned chalice or the holy grail. Inf. Software Technol. **47**(4), 233–244 (2005). https://doi.org/10.1016/j.infsof.2004.08.005
176. Miller, J., Wood, M., Roper, M.: Further experiences with scenarios and checklists. Empirical Software Eng. **3**(1), 37–64 (1998). https://doi.org/10.1023/A:1009735805377
177. Mills, H.D., Dyer, M., Linger, R.C.: Cleanroom software engineering. IEEE Software **4**(05), 19–25 (1987). https://doi.org/10.1109/MS.1987.231413
178. Moe, N.B., Aurum, A., Dybå, T.: Challenges of shared decision-making: a multiple case study of agile software development. Inf. Software Technol. **54**(8), 853–865 (2012). https://doi.org/10.1016/j.infsof.2011.11.006
179. Molléri, J.S., Petersen, K., Mendes, E.: Survey guidelines in software engineering: an annotated review. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 1–6 (2016). https://doi.org/10.1145/2961111.2962619
180. Montgomery, D.C.: Design and Analysis of Experiments, 10th edn. John Wiley & Sons, Hoboken (2019)
181. Mourão, E., Kalinowski, M., Murta, L., Mendes, E., Wohlin, C.: Investigating the use of a hybrid search strategy for systematic reviews. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 193–198 (2017). https://doi.org/10.1109/ESEM.2017.30

182. Mourão, E., Pimentel, J.F., Murta, L., Kalinowski, M., Mendes, E., Wohlin, C.: On the performance of hybrid search strategies for systematic literature reviews in software engineering. Inf. Software Technol. **123**, 106294 (2020). https://doi.org/10.1016/j.infsof.2020.106294
183. Müller, M., Pfahl, D.: Simulation methods. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering, pp. 117–152. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_5
184. Myers, G.J.: A controlled experiment in program testing and code walkthroughs/inspections. Commun. ACM **21**, 760–768 (1978). https://doi.org/10.1145/359588.359602
185. Neto, G.T.G., Santos, W.B., Endo, P.T., Fagundes, R.A.A.: Multivocal literature reviews in software engineering: preliminary findings from a tertiary study. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 1–6 (2019). https://doi.org/10.1109/ESEM.2019.8870142
186. Noblit, G.W., Hare, R.D.: Meta-Ethnography: Synthesizing Qualitative Studies. Sage Publications, Thousand Oaks (1988)
187. Ohlsson, M.C., Wohlin, C.: A project effort estimation study. Inf. Software Technol. **40**(14), 831–839 (1998). https://doi.org/10.1016/S0950-5849(98)00097-4
188. Owen, S., Brereton, P., Budgen, D.: Protocol analysis: a neglected practice. Commun. ACM **49**(2), 117–122 (2006). https://doi.org/10.1145/1113034.1113039
189. Pérez, J., Díaz, J., Garcia-Martin, J., Tabuenca, B.: Systematic literature reviews in software engineering—enhancement of the study selection process using cohen's kappa statistic. J. Syst. Software **168**, 110657 (2020). https://doi.org/10.1016/j.jss.2020.110657
190. Petersen, K., Wohlin, C.: Context in industrial software engineering research. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement, pp. 401–404 (2009). https://doi.org/10.1109/ESEM.2009.5316010
191. Petersen, K., Wohlin, C.: The effect of moving from a plan-driven to an incremental and agile development approach: an industrial case study. J. Empirical Software Eng. **15**(6), 654–693 (2010). https://doi.org/10.1007/s10664-010-9136-6
192. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (2008). https://doi.org/10.14236/ewic/EASE2008.8
193. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Software Technol. **64**, 1–18 (2015). https://doi.org/10.1016/j.infsof.2015.03.007
194. Pfleeger, S.L.: Experimental design and analysis in software engineering. Ann. Software Eng. **1**, 219–253 (1995)
195. Pfleeger, S.L., Atlee, J.M.: Software Engineering: Theory and Practice, 4th edn. Pearson Prentice-Hall, Hoboken (2009)
196. Pickard, L.M., Kitchenham, B.A., Jones, P.W.: Combining empirical results in software engineering. Inf. Software Technol. **40**(14), 811–821 (1998). https://doi.org/10.1016/S0950-5849(98)00101-3
197. Pinsonneault, A., Kraemer, K.: Survey research methodology in management information systems: an assessment. J. Manage. Inf. Syst., 75–105 (1993). https://doi.org/10.1080/07421222.1993.11518001
198. Porter, A.A., Votta, L.G.: An experiment to assess different defect detection methods for software requirements inspections. In: Proceedings of the International Conference on Software Engineering, pp. 103–112 (1994). https://doi.org/10.1109/ICSE.1994.296770
199. Porter, A., Votta, L.: Comparing detection methods for software requirements inspection: a replication using professional subjects. Empirical Software Eng. **3**(4), 355–380 (1998). https://doi.org/10.1023/A:1009776104355
200. Porter, A.A., Votta, L.G., Basili, V.R.: Comparing detection methods for software requirements inspection: a replicated experiment. IEEE Trans. Software Eng. **21**(6), 563–575 (1995). https://doi.org/10.1109/32.391380

201. Porter, A.A., Siy, H.P., Toman, C.A., Votta, L.G.: An experiment to assess the cost-benefits of code inspections in large scale software development. IEEE Trans. Software Eng. **23**(6), 329–346 (1997). https://doi.org/10.1109/32.601071
202. Potts, C.: Software engineering research revisited. IEEE Software, 19–28 (1993). https://doi.org/10.1109/52.232392
203. Rainer, A.: The longitudinal, chronological case study research strategy: a definition, and an example from IBM Hursley Park. Inf. Software Technol. **53**(7), 730–746 (2011). https://doi.org/10.1016/j.infsof.2011.01.003
204. Ralph, P.: Empirical standards for software engineering research – case study and ethnography. https://acmsigsoft.github.io/EmpiricalStandards/docs/?standard=CaseStudy (2021). This is an online resource, edited by P. Ralph
205. Robinson, H., Segal, J., Sharp, H.: Ethnographically-informed empirical studies of software practice. Inf. Software Technol. **49**(6), 540–551 (2007). https://doi.org/10.1016/j.infsof.2007.02.007
206. Robson, C.: Small-Scale Evaluation: Principles and Practice. Sage Publications Ltd., Thousand Oaks (2017)
207. Robson, C., McCartan, K.: Real World Research: A Resource for Users of Social Research Methods in Applied Settings, 4th edn. Wiley, Hoboken (2016)
208. Ros, R., Bjarnason, E., Runeson, P.: A machine learning approach for semi-automated search and selection in literature studies. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, pp. 118–127 (2017). https://doi.org/10.1145/3084226.3084243
209. Ros, R., Bjarnason, E., Runeson, P.: A theory of factors affecting continuous experimentation (FACE). Empirical Software Eng. **29**(21) (2024). https://doi.org/10.1007/s10664-023-10358-z
210. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Eng. **14**, 131–164 (2009). https://doi.org/10.1007/s10664-008-9102-8
211. Runeson, P., Skoglund, M.: Reference-based search strategies in systematic reviews. In: Proceedings of the International Conference on Empirical Assessment & Evaluation in Software Engineering (2009). https://doi.org/10.14236/ewic/EASE2009.4
212. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case Study Research in Software Engineering. Guidelines and Examples. John Wiley & Sons, Hoboken (2012)
213. Runeson, P., Engström, E., Storey, M.A.: The design science paradigm as a frame for empirical software engineering. In: Felderer, M., Travassos, G.H. (eds.) Contemporary Empirical Methods in Software Engineering, pp. 127–147. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-32489-6_5
214. Runeson, P., Söderberg, E., Höst, M.: A conceptual framework and recommendations for open data and artifacts in empirical software engineering. In: Proceedings of the International Workshop on Methodological Issues with Empirical Studies in Software Engineering (2024). https://doi.org/10.1145/3643664.3648206
215. Sandahl, K., Blomkvist, O., Karlsson, J., Krysander, C., Lindvall, M., Ohlsson, N.: An extended replication of an experiment for assessing methods for software requirements. Empirical Software Eng. **3**(4), 381–406 (1998). https://doi.org/10.1023/A:1009724120285
216. Santos, A., Gómez, O., Juristo, N.: Analyzing families of experiments in SE: a systematic mapping study. IEEE Trans. Software Eng. **46**(5), 566–583 (2020). https://doi.org/10.1109/TSE.2018.2864633
217. Seaman, C.B.: Qualitative methods in empirical studies of software engineering. IEEE Trans. Software Eng. **25**(4), 557–572 (1999). https://doi.org/10.1109/32.799955
218. Selby, R.W., Basili, V.R., Baker, F.T.: Cleanroom software development: an empirical evaluation. IEEE Trans. Software Eng. **13**(9), 1027–1037 (1987). https://doi.org/10.1109/TSE.1987.233525

219. Sharp, H., Dittrich, Y., De Souza, C.R.B.: The role of ethnographic studies in empirical software engineering. IEEE Trans. Software Eng. **42**(8), 786–804 (2016). https://doi.org/10.1109/TSE.2016.2519887
220. Shepperd, M.: Foundations of Software Measurement. Prentice-Hall International, Hoboken (1995)
221. Shepperd, M., Ajienka, N., Counsell, S.: The role and value of replication in empirical software engineering results. Inf. Software Technol. **99**, 120–132 (2018). https://doi.org/10.1016/j.infsof.2018.01.006
222. Shneiderman, B., Mayer, R., McKay, D., Heller, P.: Experimental investigations of the utility of detailed flowcharts in programming. Commun. ACM **20**, 373–381 (1977). https://doi.org/10.1145/359605.359610
223. Shull, F.J.: Developing techniques for using software documents: a series of empirical studies. Ph.D. Thesis, Computer Science Department, University of Maryland (1998)
224. Shull, F., Basili, V., Carver, J., Maldonado, J.C., Travassos, G.H., Mendonça, M.G., Fabbri, S.: Replicating software engineering experiments: addressing the tacit knowledge problem. In: Proceedings of the International Symposium on Empirical Software Engineering, pp. 7–16 (2002). https://doi.org/10.1109/ISESE.2002.1166920
225. Shull, F., Mendoncça, M.G., Basili, V., Carver, J., Maldonado, J.C., Fabbri, S., Travassos, G.H., Ferreira, M.C.: Knowledge-sharing issues in experimental software engineering. Empirical Software Eng. **9**, 111–137 (2004). https://doi.org/10.1023/B:EMSE.0000013516.80487.33
226. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Eng. **13**, 211–218 (2008). https://doi.org/10.1007/s10664-008-9060-1
227. Sieber, J.E.: Protecting research subjects, employees and researchers: implications for software engineering. Empirical Software Eng. **6**(4), 329–341 (2001). https://doi.org/10.1023/A:1011978700481
228. Siegel, S., Castellan, J.: Nonparametric Statistics for the Behavioral Sciences, 2nd edn. McGraw-Hill International Editions, New York (1988)
229. Siegmund, J., Siegmund, N., Apel, S.: Views on internal and external validity in empirical software engineering. In: Proceedings of the International Conference on Software Engineering, pp. 9–19 (2015). https://doi.org/10.1109/ICSE.2015.24
230. Singer, J., Vinson, N.: Why and how research ethics matters to you. Yes, you! Empirical Software Eng. **6**, 287–290 (2001). https://doi.org/10.1023/A:1011998412776
231. Singer, J., Vinson, N.G.: Ethical issues in empirical studies of software engineering. IEEE Trans. Software Eng. **28**(12), 1171–1180 (2002). https://doi.org/10.1109/TSE.2002.1158289
232. Singh, S.: Fermat's Last Theorem. Fourth Estate, London (1997)
233. Sjøberg, D.I.K., Bergersen, G.R.: Construct validity in software engineering. IEEE Trans. Software Eng. **49**(3), 1374–1396 (2023). https://doi.org/10.1109/tse.2022.3176725
234. Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.K., Rekdal, A.C.: A survey of controlled experiments in software engineering. IEEE Trans. Software Eng. **31**(9), 733–753 (2005). https://doi.org/10.1109/TSE.2005.97
235. Sjøberg, D.I.K., Dybå, T., Jørgensen, M.: The future of empirical methods in software engineering research. In: Proceedings of the Future of Software Engineering, pp. 358–378 (2007). https://doi.org/10.1109/FOSE.2007.30
236. Sjøberg, D.I.K., Dybå, T., Anda, B.C.D., Hannay, J.E.: Building theories in software engineering. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_12
237. Solari, M., Vegas, S., Juristo, N.: Content and structure of laboratory packages for software engineering experiments. Inf. Software Technol. **97**, 64–79 (2018). https://doi.org/10.1016/j.infsof.2017.12.016
238. Sommerville, I.: Software Engineering, 9th edn. Addison-Wesley, Boston (2011)

239. Sørumgård, S.: Verification of process conformance in empirical studies of software development. Ph.D. thesis, The Norwegian University of Science and Technology, Department of Computer and Information Science (1997)
240. Stake, R.E.: The Art of Case Study Research. SAGE Publications, Thousand Oaks (1995)
241. Staples, M., Niazi, M.: Experiences using systematic review guidelines. J. Syst. Software **80**(9), 1425–1437 (2007). https://doi.org/10.1016/j.jss.2006.09.046
242. Staron, M.: Action Research in Software Engineering. Springer International Publishing, Berlin (2020)
243. Staron, M., Kuzniarz, L., Wohlin, C.: Empirical assessment of using stereotypes to improve comprehension of UML models: a set of experiments. J. Syst. Software **79**(5), 727–742 (2006). https://doi.org/10.1016/j.jss.2005.09.014
244. Stavru, S.: A critical examination of recent industrial surveys on agile method usage. J. Syst. Software **94**, 87–97 (2014). https://doi.org/10.1016/j.jss.2014.03.041
245. Stol, K.J., Fitzgerald, B.: Theory-oriented software engineering. Sci. Comput. Program. **101**, 79–98 (2015). https://doi.org/10.1016/j.scico.2014.11.010
246. Stol, K.J., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: Proceedings of the International Conference on Software Engineering, pp. 120–131 (2016). https://doi.org/10.1145/2884781.2884833
247. Storey, M.A., Ernst, N.A., Williams, C., Kalliamvakou, E.: The who, what, how of software engineering research: a socio-technical framework. Empirical Software Eng. **25**(5), 4097–4129 (2020). https://doi.org/10.1007/s10664-020-09858-z
248. Strandberg, P.E.: Ethical interviews in software engineering. In: International Symposium on Empirical Software Engineering and Measurement, pp. 1–11 (2019). https://doi.org/10.1109/ESEM.2019.8870192
249. Thelin, T., Runeson, P.: Capture-recapture estimations for perspective-based reading – a simulated experiment. In: Proceedings of the International Conference on Product Focused Software Process Improvement, Oulu, pp. 182–200 (1999)
250. Thelin, T., Runeson, P., Wohlin, C.: An experimental comparison of usage-based and checklist-based reading. IEEE Trans. Software Eng. **29**(8), 687–704 (2003). https://doi.org/10.1109/TSE.2003.1223644
251. Tichy, W.F.: Should computer scientists experiment more? IEEE Comput. **31**(5), 32–39 (1998). https://doi.org/10.1109/2.675631
252. Tichy, W.F., Lukowicz, P., Prechelt, L., Heinz, E.A.: Experimental evaluation in computer science: a quantitative study. J. Syst. Software **28**(1), 9–18 (1995). https://doi.org/10.1016/0164-1212(94)00111-Y
253. Trochim, W.M.K., Donnelly, J.P., Arora, K.: Research Methods: The Essential Knowledge Base. Cengage Learning, Boston (2016)
254. UNESCO: Understanding open science. Technical Report. SC-PBS-STIP/2022/OST/1, UNESCO (2022). https://doi.org/10.54677/UTCD9302
255. Usman, M., bin Ali, N., Wohlin, C.: A quality assessment instrument for systematic literature reviews in software engineering. e-Inform. Software Eng. J. **17**(1), 230105 (2023). https://doi.org/10.37190/e-Inf230105
256. van Dinter, R., Tekinerdogan, B., Catal, C.: Automation of systematic literature reviews: a systematic literature review. Inf. Software Technol. **136**, 106589 (2021). https://doi.org/10.1016/j.infsof.2021.106589
257. van Solingen, R., Berghout, E.: The Goal/Question/Metric Method: A Practical Guide for Quality Improvement and Software Development. McGraw-Hill International, New York (1999)
258. van Vliet, H.: Software Engineering: Principles and Practice, vol. 13. John Wiley & Sons, Hoboken (2008)
259. Verdecchia, R., Engström, E., Lago, P., Runeson, P., Song, Q.: Threats to validity in software engineering research: a critical reflection. Inf. Software Technol., 107329 (2023). https://doi.org/10.1016/j.infsof.2023.107329

260. Verner, J.M., Sampson, J., Tosic, V., Abu Bakar, N.A., Kitchenham, B.A.: Guidelines for industrially-based multiple case studies in software engineering. In: Proceedings of the International Conference on Research Challenges in Information Science, pp. 313–324 (2009). https://doi.org/10.1109/RCIS.2009.5089295
261. Vidoni, M.: A systematic process for mining software repositories: results from a systematic literature review. Inf. Software Technol. **144**, 106791 (2022). https://doi.org/10.1016/j.infsof.2021.106791
262. Vinson, N.G., Singer, J.: A practical guide to ethical research involving humans. In: Shull, F., Singer, J., Sjøberg, D.I.K. (eds.) Guide to Advanced Empirical Software Engineering. Springer, London (2008). https://doi.org/10.1007/978-1-84800-044-5_9
263. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al.: SciPy 1.0: fundamental algorithms for scientific computing in python. Nat. Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
264. Votta, L.G.: Does every inspection need a meeting? ACM Software Eng. Notes **18**(5), 107–114 (1993). https://doi.org/10.1145/167049.167070
265. Wallace, C., Cook, C., Summet, J., Burnett, M.: Assertions in end-user software engineering: a think-aloud study. In: Proceedings of the Symposia on Human Centric Computing Languages and Environments, pp. 63–65 (2002). https://doi.org/10.1109/HCC.2002.1046348
266. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer, Berlin (2014). https://doi.org/10.1007/978-3-662-43839-8
267. Williams, L.A., Kessler, R.R.: Experiments with industry's "pair-programming" model in the computer science classroom. Comput. Sci. Educ. **11**(1), 7–20 (2001). https://doi.org/10.1076/csed.11.1.7.3846
268. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (2014). https://doi.org/10.1145/2601248.2601268
269. Wohlin, C.: Case study research in software engineering—it is a case, and it is a study, but is it a case study? Inf. Software Technol. **133**, 106514 (2021). https://doi.org/10.1016/j.infsof.2021.106514
270. Wohlin, C., Aurum, A.: Towards a decision-making structure for selecting a research design in empirical software engineering. Empirical Software Eng. **20**(6), 1427–1455 (2015). https://doi.org/10.1007/s10664-014-9319-7
271. Wohlin, C., Rainer, A.: Is it a case study?–a critical analysis and guidance. J. Syst. Software **192**, 111395 (2022). https://doi.org/10.1016/j.jss.2022.111395
272. Wohlin, C., Runeson, P.: Guiding the selection of research methodology in industry–academia collaboration in software engineering. Inf. Software Technol. **140**, 106678 (2021). https://doi.org/10.1016/j.infsof.2021.106678
273. Wohlin, C., Gustavsson, A., Höst, M., Mattsson, C.: A framework for technology introduction in software organizations. In: Proceedings of the Conference on Software Process Improvement, pp. 167–176 (1996)
274. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering – An Introduction. Kluwer Academic Publisher, Dordrecht (2000)
275. Wohlin, C., Aurum, A., Angelis, L., Phillips, L., Dittrich, Y., Gorschek, T., Grahn, H., Henningsson, K., Kågström, S., Low, G., Rovegård, P., Tomaszewski, P., van Toorn, C., Winter, J.: Success factors powering industry-academia collaboration in software research. IEEE Software **29**(2), 67–73 (2012). https://doi.org/10.1109/MS.2011.92
276. Wohlin, C., Šmite, D., Moe, N.B.: A general theory of software engineering: balancing human, social and organizational capitals. J. Syst. Software **109**, 229–242 (2015). https://doi.org/10.1016/j.jss.2015.08.009
277. Wohlin, C., Mendes, E., Romero Felizardo, K., Kalinowski, M.: Guidelines for the search strategy to update systematic literature reviews in software engineering. Inf. Software Technol. **127**, 106366 (2020). https://doi.org/10.1016/j.infsof.2020.106366

278. Wohlin, C., Kalinowski, M., Romero Felizardo, K., Mendes, E.: Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. Inf. Software Technol. **147**, 106908 (2022). https://doi.org/10.1016/j.infsof.2022.106908
279. Yang, M., Nazir, S., Xu, Q., Ali, S.: Deep learning algorithms and multicriteria decision-making used in big data: a systematic literature review. Complexity **2020**, 2836064 (2020). https://doi.org/10.1155/2020/2836064
280. Yang, L., Zhang, H., Shen, H., Huang, X., Zhou, X., Rong, G., Shao, D.: Quality assessment in systematic literature reviews: a software engineering perspective. Inf. Software Technol. **130**, 106397 (2021). https://doi.org/10.1016/j.infsof.2020.106397
281. Yin, R.K.: Case Study Research Design and Methods, 4th edn. Sage Publications, Beverly Hills (2009)
282. Zain, Z.M., Sakri, S., Ismail, N.H.A.: Application of deep learning in software defect prediction: systematic literature review and meta-analysis. Inf. Software Technol. **158**, 107175 (2023). https://doi.org/10.1016/j.infsof.2023.107175
283. Zannier, C., Melnik, G., Maurer, F.: On the success of empirical studies in the international conference on software engineering. In: Proceedings of the International Conference on Software Engineering, pp. 341–350 (2006). https://doi.org/10.1145/1134285.1134333
284. Zelkowitz, M.V., Wallace, D.R.: Experimental models for validating technology. IEEE Comput. **31**(5), 23–31 (1998). https://doi.org/10.1109/2.675630
285. Zendler, A.: A preliminary software engineering theory as investigated by published experiments. Empirical Software Eng. **6**, 161–180 (2001). https://doi.org/10.1023/A:1011489321999
286. Zhang, L., Tian, J.H., Jiang, J., Liu, Y.J., Pu, M.Y., Yue, T.: Empirical research in software engineering–a literature survey. J. Comput. Sci. Technol. **33**, 876–899 (2018). https://doi.org/10.1007/s11390-018-1864-x

Index

C. Wohlin et al., *Experimentation in Software Engineering*,
https://doi.org/10.1007/978-3-662-69306-3